

TOTALVIEW

GRAPHIC USER

INTERFACE

COMMANDS GUIDE



FEBRUARY 2005

VERSION 6.7

Copyright © 1998–2005 by Etnus LLC. All rights reserved.

Copyright © 1996–1998 by Dolphin Interconnect Solutions, Inc.

Copyright © 1993–1996 by BBN Systems and Technologies, a division of BBN Corporation.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise without the prior written permission of Etnus LLC. (Etnus).

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Etnus has prepared this manual for the exclusive use of its customers, personnel, and licensees. The information in this manual is subject to change without notice, and should not be construed as a commitment by Etnus. Etnus assumes no responsibility for any errors that appear in this document.

TotalView and Etnus are registered trademarks of Etnus LLC.

TotalView uses a modified version of the Microline widget library. Under the terms of its license, you are entitled to use these modifications. The source code is available at <http://www.etnus.com/Products/TotalView/developers>.

All other brand names are the trademarks of their respective holders.



Book Overview



window 1	The Root Window	1
window 2	The Process Window	41
window 3	The Variable Window	121
window 4	The Visualizer Window	145
window 5	Fortran Modules Window	149
window 6	Program Browser Window	155
window 7	Message Queue Window	161
window 8	PVM Tasks Window	169
window 9	Memory Debugging Window	175
window 10	Thread Objects Window	207
window 11	Expression List Window	219
window 12	Other Topics	229

Contents

window 1 The Root Window

Root Window Pages	1
Attached Page	1
Unattached Page	3
Groups Page	5
Log Page	5
File Menu Commands	6
File > New Program	6
File > Search Path	8
File > Preferences	9
Options Page	9
Action Points Page	12
Launch Strings Page	14
Single Debug Server Launch	14
Visualizer Launch	16
Source Code Editor	17
Bulk Launch Page	18
Dynamic Libraries Page	21
Parallel Page	24
Fonts Page	26
Formatting Page	27
Pointer Dive Page	29
File > Save Pane	30
File > Exit	31
Edit Menu Commands	31
Edit > Undo	31
Edit > Cut	31
Edit > Copy	32
Edit > Paste	32
Edit > Delete	32
Edit > Find	32
Edit > Find Again	33
View Menu Commands	33
View > Dive	33

View > Dive in New Window	33
View > Expand All	34
View > Collapse All	34
View > Display Manager Threads	34
View > Display Exited Threads	34
Tools Menu Commands	34
Tools > Restart Checkpoint	34
Tools > P/T Set Browser	36
Tools > PVM Tasks	37
Tools > Memory Debugging	37
Tools > Command Line	37
Window Menu Commands	37
Window > Update	38
Window > Update All	38
Window > Memorize	38
Window > Memorize all	38

window 2 The Process Window

Process Window Panes	41
Stack Trace Pane	41
Stack Frame Pane	42
Source Pane	42
Threads Pane	43
Action Points Pane	43
File Menu Commands	44
File > New Program	44
File > Search Path	46
File > Signals	47
File > Preferences	48
File > Open Source	48
File > Edit Source	49
File > Save Pane	49
File > Rescan Libraries	50
File > Close Relatives	50
File > Close	50
File > Exit	50
Edit Menu Commands	51
Edit > Undo	51
Edit > Cut	51
Edit > Copy	51
Edit > Paste	51
Edit > Delete	51
Edit > Find	52
Edit > Find Again	52
View Menu Commands	53
View > Dive	53
View > Dive in New Window	54
View > Undive	54
View > Redive	54
View > Reset	54
View > Lookup Function	55

View > Lookup Variable	55
View > Next Process	57
View > Previous Process	57
View > Next Thread	57
View > Previous Thread	57
View > Source As > Source	57
View > Source As > Assembler	57
View > Source As > Both	57
View > Assembler > Symbolically	58
View > Assembler > By Address	58
Group Menu Commands	59
Group > Go	59
Group > Halt	60
Group > Next	60
Group > Step	60
Group > Out	61
Group > Run To	61
Group > Next Instruction	62
Group > Step Instruction	62
Group > Share Submenu	63
Group > Share > Go	63
Group > Share > Halt	63
Group > Share > Next	64
Group > Share > Step	64
Group > Share > Out	64
Group > Share > Run To	65
Group > Share > Next Instruction	65
Group > Share > Step Instruction	66
Group > Workers Submenu	66
Group > Workers > Go	66
Group > Workers > Halt	66
Group > Workers > Next	67
Group > Workers > Step	67
Group > Workers > Out	67
Group > Workers > Run To	68
Group > Workers > Next Instruction	69
Group > Workers > Step Instruction	69
Group > Lockstep Submenu	69
Group > Lockstep > Go	70
Group > Lockstep > Halt	70
Group > Lockstep > Next	70
Group > Lockstep > Step	70
Group > Lockstep > Out	70
Group > Lockstep > Run To	71
Group > Lockstep > Next Instruction	71
Group > Lockstep > Step Instruction	71
Group > Hold	72
Group > Release	72
Group > Attach Subset	72
Group > Edit Group	74
Group > Restart	76

Group > Delete	76
Process Menu Commands	76
Process > Go	77
Process > Halt	77
Process > Next	77
Process > Step	77
Process > Out	78
Process > Run To	78
Process > Next Instruction	79
Process > Step Instruction	79
Process > Workers Submenu	80
Process > Workers > Go	80
Process > Workers > Halt	80
Process > Workers > Next	80
Process > Workers > Step	81
Process > Workers > Out	81
Process > Workers > Run To	82
Process > Workers > Next Instruction	82
Process > Workers > Step Instruction	82
Process > Lockstep Submenu	83
Process > Lockstep > Go	83
Process > Lockstep > Halt	83
Process > Lockstep > Next	83
Process > Lockstep > Step	84
Process > Lockstep > Out	84
Process > Lockstep > Run To	84
Process > Lockstep > Next Instruction	85
Process > Lockstep > Step Instruction	85
Process > Hold	86
Process > Hold Threads	86
Process > Release Threads	86
Process > Create	87
Process > Detach	87
Process > Startup Parameters	87
Arguments Page	87
Environment Page	88
Standard I/O Page	89
Thread Menu Commands	91
Thread > Go	91
Thread > Halt	91
Thread > Next	91
Thread > Step	92
Thread > Out	92
Thread > Run To	93
Thread > Next Instruction	93
Thread > Step Instruction	94
Thread > Set PC	94
Thread > Hold	94
Thread > Continuation Signal	94
Action Point Menu	95
Action Point > Set Breakpoint	95

Action Point > Set Barrier	96
Action Point > At Location	96
Action Point > Enable	96
Action Point > Disable	96
Action Point > Delete	96
Action Point > Properties	96
Action Point > Suppress All	102
Action Point > Delete All	102
Action Point > Load All	102
Action Point > Save All	103
Action Point > Save As	103
Tools Menu Commands	103
Tools > Evaluate	103
Tools > Expression List	105
Tools > Program Browser	105
Tools > Fortran Modules	105
Tools > Call Tree	105
Tools > Debugger Loaded Libraries	106
Tools > Memory Debugging	107
Tools > Memory Block Properties	107
Tools > Memory Event Details	109
Tools > Thread Objects	109
Tools > Message Queue	110
Tools > Message Queue Graph	110
Tools > Create Checkpoint	112
Tools > Restart Checkpoint	115
Tools > PVM Tasks	117
Tools > Global Arrays	117
Tools > Command Line	117
Window Menu Commands	117
Window > Update	118
Window > Update All	118
Window > Duplicate	118
Window > Memorize	118
Window > Memorize all	118
Window > Root	119

window 3 The Variable Window

Variable Window Overview	121
File Menu Commands	126
File > Save Pane	126
File > Close Similar	126
File > Close	127
File > Exit	127
Edit Menu Commands	127
Edit > Undo	127
Edit > Reset Defaults	127
Edit > Cut	127
Edit > Copy	128
Edit > Paste	128
Edit > Delete	128

Edit > Find	128
Edit > Find Again	129
View Menu Commands	129
View > Dive	129
View > Dive in New Window	130
View > Dive In All	130
View > Expand All	130
View > Collapse All	130
View > Add to Expression List	130
View > Undive	130
View > Undive All	131
View > Redive	131
View > Redive All	131
View > Laminate > None	131
View > Laminate > Process	131
View > Laminate > Thread	131
View > Compilation Scope > Fixed	132
View > Compilation Scope > Floating	132
View > Loader Symbols	132
View > Padding	132
Tools Menu Commands	133
Tools > Create Watchpoint	133
Tools > Add to Expression List	136
Tools > Block Properties	136
Tools > Visualize	138
Tools > Visualize Distribution	138
Tools > Statistics	138
Tools > Attach Subset (Array of Ranks)	140
Window Menu Commands	142
Window > Update	142
Window > Update All	142
Window > Duplicate	142
Window > Memorize	142
Window > Memorize all	142
Window > Root	143
Variable Details Window	143

window 4 The Visualizer Window

Directory Window	146
File Menu Commands	146
View Menu Commands	146
Options Menu Command	146
Data Window	146
File Menu Commands	147

window 5 Fortran Modules Window

File Menu Commands	150
File > Close Similar	150
File > Close	150
Edit Menu Commands	150
Edit > Undo	150
Edit > Cut	150

Edit > Copy	151
Edit > Paste	151
Edit > Delete	151
Edit > Find	151
Edit > Find Again	152
View Menu Commands	152
View > Dive	152
View > Dive in New Window	152
Window Menu Commands	153
Window > Update	153
Window > Update All	153
Window > Memorize	153
Window > Memorize all	153
Window > Root	153

window 6 **Program Browser Window**

File Menu Commands	155
File > Close Similar	155
File > Close	155
File > Exit	155
Edit Menu Commands	157
Edit > Undo	157
Edit > Cut	157
Edit > Copy	157
Edit > Paste	157
Edit > Delete	157
Edit > Find	158
Edit > Find Again	158
View Menu Commands	159
View > Dive	159
View > Dive in New Window	159
Window Menu Commands	159
Window > Update	159
Window > Update All	159
Window > Memorize	159
Window > Memorize all	160
Window > Root	160

window 7 **Message Queue Window**

Message Queue Window Overview	161
Message Operations	163
File Menu Commands	163
File > Close Similar	163
File > Close	163
File > Exit	164
Edit Menu Commands	164
Edit > Undo	164
Edit > Cut	164
Edit > Copy	164
Edit > Paste	165
Edit > Delete	165
Edit > Find	165

Edit > Find Again	166
View Menu Commands	166
View > Dive	166
View > Dive in New Window	166
Window Menu Commands	167
Window > Update	167
Window > Update All	167
Window > Memorize	167
Window > Memorize all	167
Window > Root	167
window 8 PVM Tasks Window	
File Menu Commands	170
File > Close	170
File > Exit	170
Edit Menu Commands	171
Edit > Undo	171
Edit > Cut	171
Edit > Copy	171
Edit > Paste	171
Edit > Delete	171
Edit > Find	172
Edit > Find Again	172
View Menu Commands	173
View > Dive	173
View > Dive in New Window	173
Window Menu Commands	173
Window > Update	173
Window > Update All	173
Window > Memorize	173
Window > Memorize all	174
Window > Root	174
window 9 Memory Debugging Window	
Memory Debugging Window Overview	175
Rows and Columns	177
Saving Views	177
Configuration Page	179
Leak Detection Page	187
Heap Status Page	191
Memory Usage Page	195
File Menu Commands	197
File > Preferences	197
File > Close	198
Edit Menu Commands	198
Edit > Copy	198
Edit > Select All	198
Edit > Find	198
Edit > Find Again	199
View Menu Commands	199
View > Collapse All	199
View > Expand All	199

Action Menu Commands	199
Action > Generate View	199
Action > View Preferences	199
Tools Menu Commands	200
Tools > Filters	200
Window Menu Commands	204
Window > Update	204
Window > Update All	205
Window > Duplicate	205
Window > Root	205
Event Types	205

window 10 Thread Objects Window

HP Tru64 UNIX	207
Mutexes Page.....	207
Condition Variables.....	209
IBM AIX	210
Mutexes Page.....	210
Condition Variables Page.....	211
R/W Locks Page	212
Data Keys Page	214
File Menu Commands	214
File > Close Similar	214
File > Close	214
File > Exit	214
Edit Menu Commands	215
Edit > Undo	215
Edit > Cut	215
Edit > Copy	215
Edit > Paste	215
Edit > Delete	215
Edit > Find	216
Edit > Find Again	216
View Menu Commands	217
View > Dive	217
View > Dive ... in New Window	217
Window Menu Commands	217
Window > Update	217
Window > Update All	218
Window > Memorize	218
Window > Memorize all	218
Window > Root	218

window 11 Expression List Window

Expression List Window Overview	219
Entering Variables and Expression into the Expression List Window	219
Opening and Closing the Expression List Window	221
What Can You Enter in the Expression Column.....	221
Manipulating the Expression List Window	221
Multiprocess/Multithreaded Behavior	222
File Menu Commands	223

File > Preferences	223
File > Save Pane	223
File > Close Similar	224
File > Close	224
File > Exit	224
Edit Menu Commands	224
Edit > Undo	224
Edit > Reset Default	224
Edit > Cut	224
Edit > Copy	225
Edit > Paste	225
Edit > Delete	225
Edit > Delete Expression	225
Edit > Delete All Expressions	225
Edit > Duplicate Expression	225
Edit > Find	226
Edit > Find Again	226
View Menu Commands	227
View > Dive	227
Window Menu Commands	227
Window > Update	227
Window > Update All	227
Window > Duplicate	227
Window > Memorize	227
Window > Memorize all	227
Window > Root	228

window 12 Other Topics

Other Dialog Boxes	229
Ambiguous Line Dialog Box	229
Ambiguous Function Dialog Box	229
Other Topics	230
MPICH	230
IBM PE	230
SGI MPI	231
RMS MPI	231
Installing tvheap_mr.a	232
LIBPATH and Linking	232

The Root Window

1

Root Window Pages

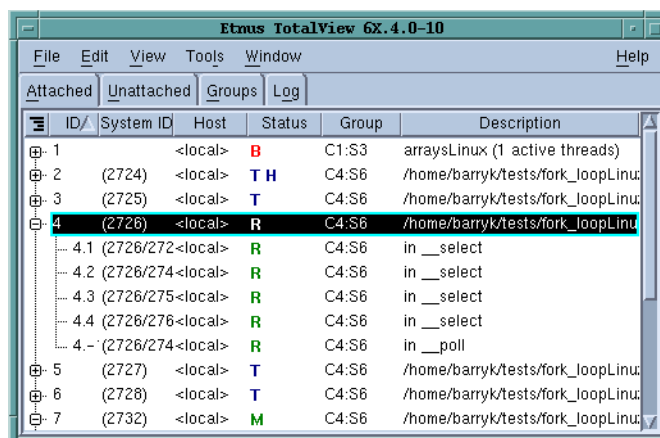
The Root Window has the following pages:


- "Attached Page" on page 1
- "Unattached Page" on page 3
- "Groups Page" on page 5
- "Log Page" on page 5

Attached Page

Contains a list of all the processes and threads you are currently debugging. Diving into an process or thread by double-clicking or using the **View > Dive** command opens a window for the selected process. If a window already exists for that process, TotalView brings it to the front. Each process is shown with its PID (Process ID) and a short summary of its state.

Figure 1: Root Window:
Showing the Attached Page



TotalView can display the Attached Page in two ways. In the preceding figure, information is shown as a "tree". Selecting the hierarchy toggle button () shifts the view from a structured to linear view. The difference

between the views is that you cannot sort and aggregate the information in linear view. Sorting and aggregating is discussed later in this section.

You can control which columns TotalView displays by right-clicking anywhere within the header line. TotalView responds by displaying a context menu listing all possible choices. If a checkbox is selected within this list, TotalView displays the column. Click on the column name within this menu to change its display status.

The default Attached Page has seven columns of information:

■ Expand/collapse indicator

This Control lets you display or hide information. Clicking on a + indicator displays hidden information. Clicking on – hides the information by collapsing the display.

■ Process and Thread IDs

A TotalView-created identifier. When displaying a thread, TotalView shows it using the thread index that it created. This number is arbitrary, so don't read anything into it.

If you dive into a Process, TotalView opens (or brings to the front) a Process Window focused on an arbitrary thread. If there is no open window associated with the process, TotalView opens a window containing the first thread in the process. If you dive into a thread, TotalView displays that thread in a Process Window

TotalView does not reuse these numbers when you restart your program or when processes and threads are deleted.

■ System ID

The process ID created by the operating system.

■ Host

The name of the computer upon which the process is executing.

■ Status

A one-letter state indicator as follows:

Character and Meaning	Definition
blank (Not begun)	<i>Process only:</i> the process has not begun running.
Bnn (Breakpoint)	<i>nn</i> is the ID of the breakpoint if it is a thread. <i>Process:</i> one or more threads are stopped at a breakpoint. <i>Thread:</i> the thread is stopped at a breakpoint. If only one thread is at a breakpoint, or all threads are at the same breakpoint, TotalView displays the ID of that breakpoint for the process as well as the thread. If the process's threads are at different breakpoints, TotalView displays a "*" for the process's breakpoint ID.
E (Error)	The Error state usually indicates that your program received a fatal signal from the operating system. Signals such as SIGSEGV , SIGBUS , and SIGFPE can indicate an error in your program. <i>Process:</i> one or more threads are in the Error state. <i>Thread:</i> stopped because of an error.

Character and Meaning	Definition
H (Held)	Either you or TotalView is holding the process of a thread. <i>Holding</i> means that the process or thread cannot run until it is released. You can explicitly release it, or TotalView can release it if the condition that caused it to be held is satisfied.
K (Kernel)	<i>Thread only</i> : the thread is executing inside the kernel (that is, it made a system call); when a thread is in the kernel, the operating system does not allow TotalView to view the thread's full state.
M (Mixed)	<i>Process only</i> : either some threads in the process are running or the process is expecting some of its threads to stop.
R (Running)	<i>Process</i> : all threads in the process are running or can run. <i>Thread</i> : running or can run.
T (Stopped)	<i>Process</i> : one or more threads are stopped, but none are in the At Breakpoint state. <i>Thread</i> : while the thread is stopped, it is not at a breakpoint and no error occurred.
W (Watchpoint)	<i>Process</i> : one or more threads are stopped at a watchpoint. <i>Thread</i> : stopped at a watchpoint.

■ Group

An identifier created by TotalView identifying the control group in which the process resides. The number used has no real meaning. However, it is an extremely useful sort key so that you can aggregate processes that are executing in the same group.

■ Description

Information about the process or thread. If this is a process description, TotalView displays summary and status information. If this is a thread description, it indicates the last place in which the PC was seen. This is the place where the thread was last stopped.

When data is being displayed hierarchically, you can perform two operations that can't be performed when data is displayed linearly. You can:

- Selectively display information using the + or – indicators.
- Sort a column by clicking on a column header.

Combined, you can more easily see information about your program. For example, if you sort the information by clicking on the **Status** header, TotalView will group all attached progress by its status such as if it is being held, at a breakpoint, and so on. When they are aggregated like this, you can also display information selectively. See Figure 2 on page 4.

Unattached Page

Contains a list of all processes. Processes to which TotalView is already attached are shown in gray. (See Figure 3 on page 4.) The processes displayed in black are not currently running under TotalView's control. To attach TotalView to any of these processes, just double-click (or dive) on the process's line in this window.

Figure 2: Root Window:
Showing the Attached Page

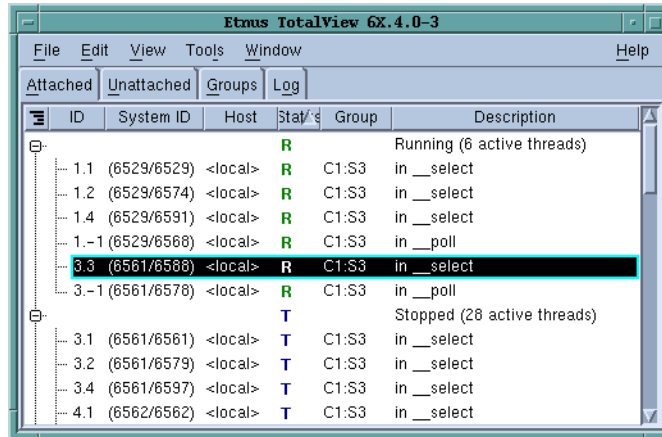
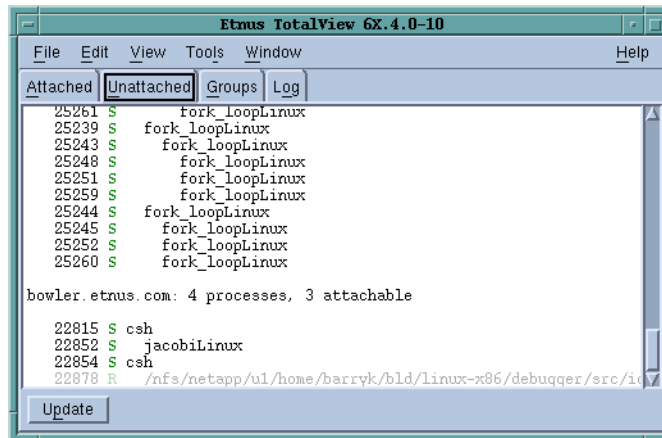


Figure 3: Root Window:
Unattached Page



If you wish to attach to a multiprocess program, you can either pick up the processes one at a time or you can restart the program under TotalView control so that the processes are automatically picked up as they are forked. In most cases, this requires you to link your program with the **dbfork** library. This is discussed in Chapter 8 of the *TotalView Reference Guide*.

If the process you are diving into is one member of a collection of related processes created with **fork()** calls, TotalView asks if you want to also attach to all of its relatives. If you answer **yes**, TotalView attaches to all the process's ancestors, descendants, and cousins.

You can control how TotalView attaches to processes by using the commands in the Parallel Page within the **File > Preferences** dialog box.



*If some of the processes in the collection have called **exec()**, TotalView tries to determine what the new executable file for the process. If TotalView appears to read the wrong file, you should start over, compile the program using the **dbfork** library, and start the program under TotalView's control.*

The information on the Unattached Page has in three columns:

- The operating system program ID.
- A letter indicating the program’s state, as follows:

Character and Meaning	Definition
I (Idle)	Process has been idle or sleeping for <i>more</i> than 20 seconds.
R (Running)	Process is running or can run.
S (Sleeping)	Process has been idle or sleeping for <i>less</i> than 20 seconds.
T (Stopped)	Process is stopped.
Z (Zombie)	Process is a "zombie"; that is, it is a child process that has terminated and is waiting for its parent process to gather its status.

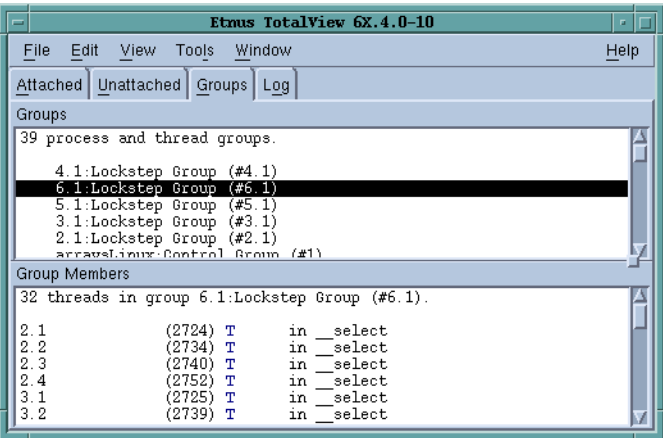
- The name of the executing process. Notice that TotalView indents names of these processes. This indentation indicates the parent/child relationship within the UNIX process hierarchy.

If you have attached to processes on more than one processor, TotalView groups this information by the processor upon which it is running.

Groups Page

Contains a window that contains a list of TotalView groups.

Figure 4: Root Window: Group Page



For information defining processes and groups, see Chapter 11, “Using Groups, Processes, and Threads” in the TotalView user Guide.

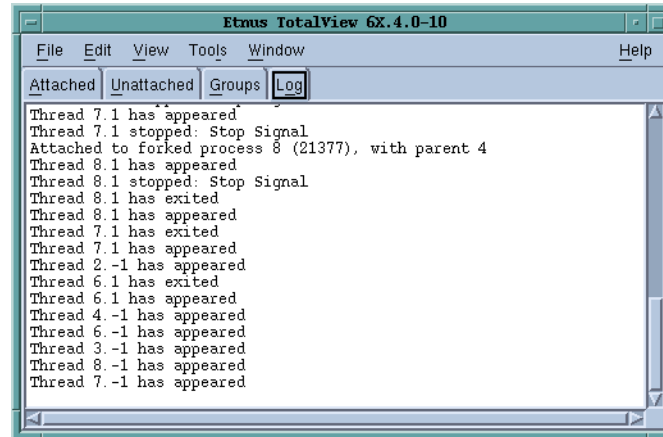
This window is divided into two parts. The top lists all the groups. The bottom shows the group members. When you select a group in the top portion, its members are displayed in the bottom portion.

Membership in groups is dynamic, changing as processes and threads are added, destroyed and execute.

Log Page

When significant events occur in the life of a process (for example, an error occurs or the process hits a breakpoint), TotalView places a line of text in the event log window indicating what occurred. (See Figure 3 on page 4.)

Figure 5: Root Window: Log Page



File Menu Commands

The commands on the **File** pulldown are:

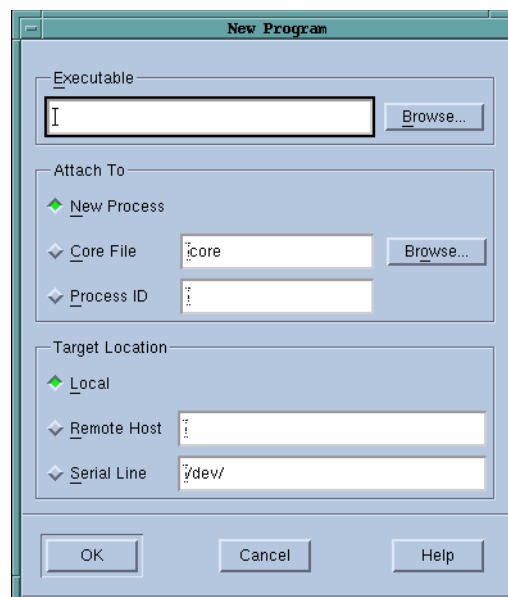
- **File > New Program** on page 6
- **File > Search Path** on page 8
- **File > Preferences** on page 9
- **File > Save Pane** on page 30
- **File > Exit** on page 31

File > New Program

Use this dialog box to specify the name of a new executable file, to attach to an existing running process or core file, or to specify the location of the process.

The **New Program** dialog box allows you to load another program.

Figure 6: File > New Program Dialog Box



When loading a program, you need to enter and indicate:

- The name of your program's executable file.
- Whether or not to attach to an existing process or core file.
- The location of the process, which can be **Local**, **Remote Host**, or **Serial Line**.

The simplest case is when you want to debug a new program on a local host. Type the name of a program you wish to debug in the **Executable** field and press **OK**.

If you want to attach to an existing process or read a core file, specify a **Process ID** or enter a name in the **Core File** field. In both of these cases, you must also enter a pathname in the **Executable** field.

If you want to debug a process on a remote machine, enter the host name or IP address of the remote machine in the **Remote Host** field.

The fields in this dialog box and their meaning are:

Executable	<p>The name of the executable file to be debugged. You can enter either a full or relative path name. If you enter just a file name, TotalView searches for the file in the directories you specified with the File > Search Path command and in all the directories named in your PATH environment variable.</p> <p>You can use the Browse button to search the file system for the file.</p>
Attach To	Lets you attach to an already running program or to load a core file.
New Process	<p>If selected, TotalView loads the executable. If the executable is already loaded, TotalView loads it again.</p>
Core file	<p>If selected, TotalView loads the core file. You must enter a program name in the Executable field because TotalView cannot know if this program is actually associated with the process.</p> <p>You can use the Browse button to search the file system for the core file.</p>
Process ID	<p>If selected, TotalView loads the program associated with this process ID. A program name must be entered in the Executable field because TotalView cannot check that this program is actually associated with the process.</p> <p>If this process is already loaded, TotalView raises the window; that is, it makes the process's window completely visible.</p> <p>If the process has children that called execve(), TotalView tries to determine each child's executable. If TotalView cannot determine the executables for the children, you need to delete (kill) the parent process and start it under TotalView control.</p>

If the executable is a multiprocess program, TotalView asks if you want to attach to all relatives of the process. To examine all processes, select **Yes**.



*This is the default behavior. You can change this behavior by using commands within the **File > Preference's Parallel Page**.*

Target Location Lets you indicate the program's location, as follows:

Local The program is on your current machine.

Remote Host

The program is on a different machine and TotalView will access it over your network.

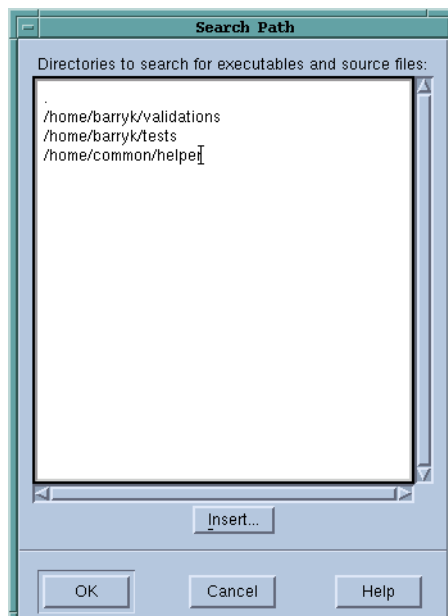
Serial Line

The program is on a different machine and TotalView will access it over a serial line.

File > Search Path

Use this dialog box to set the directories in which TotalView will search for executable and source files.

Figure 7: File > Search Path Dialog Box



You can type a directory name within the text edit box and you can use the **Insert** button to graphically move through your system's file system to select a directory to be inserted.

TotalView searches for source files, in the following order:

- 1 The current working directory (.).
- 2 The directories you specify by using the **File > Search Path** command in the exact order you enter them.
- 3 If you entered a full path name for the executable when you started TotalView, TotalView searches this directory.

- 4 If your executable is a symbolic link, TotalView will look in the directory in which your executable actually resides for the new file.
As you can have multiple levels of symbolic links, TotalView keeps on following links until it finds the actual file. After it has found the current executable, it will look in its directory for your file. If it isn't there, it'll back up the chain of links until either it finds the file or determines that the file can't be found.
- 5 The directories specified in your **PATH** environment variable.



The search path is local to the machine upon which TotalView is running. TotalView does not search for files on remote hosts.

File > Preferences

Use this dialog box to set preferences for how TotalView will behave situations, as well as define some general characteristics. The pages in this dialog box are:

- "Options Page" on page 9
- "Action Points Page" on page 12
- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

When you save your preferences, TotalView writes them to a file named **preferences6.tvd** in your **.totalview** directory. If this file exists, TotalView reads it and sets the preferences indicated within it before it begins executing.

Some preferences can differ from platform to platform. For example, the bulk launch parameters on SGI and IBM differ. Consequently, if a parameter can differ, TotalView stores a unique version for each platform. This could be the reason that a preference you set on one platform does not appear when viewing preferences on another. In general, this applies to the server launch strings and dynamic library paths.

Options Page

This page contains preferences that are basically unrelated to one another. (See Figure 8 on page 10.)

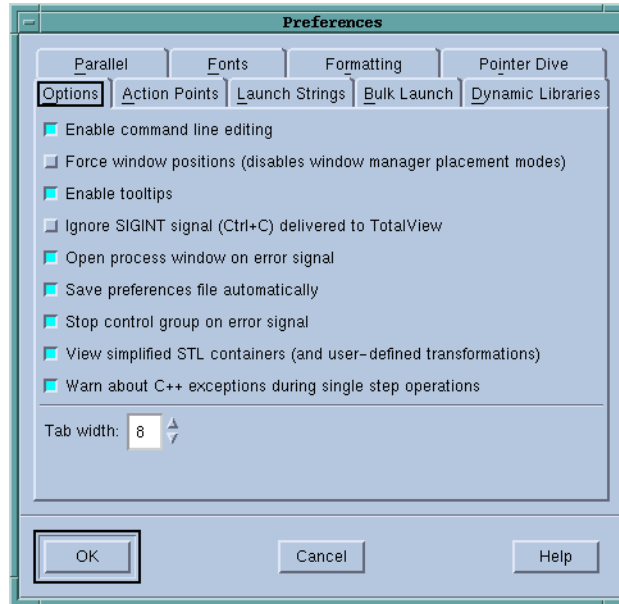
You can set the following preferences:

Enable command line editing

When set, TotalView enables some EMACS-like commands within the CLI. These commands are listed within the **dset** discussion with the CLI help.

For more information, see the **COMMAND_EDITING** variable.

Figure 8: Options Page



Force window positions (disables window manager placement modes)

Setting this preference tells TotalView that it should use the version 4 window layout algorithm. This algorithm tells the window manager where to set the window. It also cascades windows from a base location for each window type. If this is not set, which is the default, newer window managers such as **kwm** or **Enlightment** can use their smart placement modes. This is usually better.

This preference interacts with the **Window > Memorize** command. If selected, TotalView remembers the window's size and position. If it isn't selected, only the size is remembered.

Dialog boxes still chase the pointer as needed and are unaffected by this setting.

For more information, see the **TV::force_window_position** variable.

Enable tooltips

Setting this preference tells TotalView that when you place the cursor over a variable or an expression, it should display the value in a small window next to the cursor.

Ignore SIGINT signal (Ctrl+C) delivered to TotalView

When set, TotalView ignores Ctrl+C. This means you cannot use Ctrl+C to terminate TotalView. Set this checkbox if your program catches **SIGINT** signals and you do not want TotalView to see them.

For more information, see the **TV::ignore_control_c**.

Open process window on error

If selected, TotalView opens or raises the Process Window when your program encounters an error signal. (This is the default.) Deselecting this checkbox tells TotalView that it should not open or raise the window.

If processes in a multiprocess program encounter an error, TotalView only opens a Process Window for the *first* process that encounters an error. This prevents the screen from filling up with Process Windows.

For more information, see the **TV::pop_on_error** variable.

Save preferences file on exit

If selected, TotalView will write changed preferences to your preferences file. This file is stored in a **.totalview** subdirectory within your home directory.

Stop control group on error signal

If selected, TotalView stops the execution of all processes in the control group when it processes a signal as an error. See **File > Signals** for more information.

For more information, see the **TV::stop_relatives_on_proc_error** variable.

View simplified STL containers (and user-defined transformations)

If selected (and the default is selected), TotalView displays STL vector, list, and maps in a logical format rather than in their actual format. In addition, if you've added your own transformations for data types, these transforms will also be used.

For more information, see the **TV::ttf** variable.

Warn about C++ exceptions during single step operations

When set, TotalView will ask you if you wish to stop a single-step operation if your program throws a C++ exception while TotalView is stepping. The process is left stopped at the C++ run time library's throw routine.

If this is not set, TotalView will not catch C++ exceptions thrown during single-step operations. If they are not caught, a step operation could lose control of the process, and let it run away.

For more information, see the **TV::warn_step_throw** variable.

Tab width

Tells TotalView what tab interval it should use when it displays tabs that are embedded within your source code.

For more information, see the **TV::source_pane_tab_width** variable.

For information on other Preference pages, see:

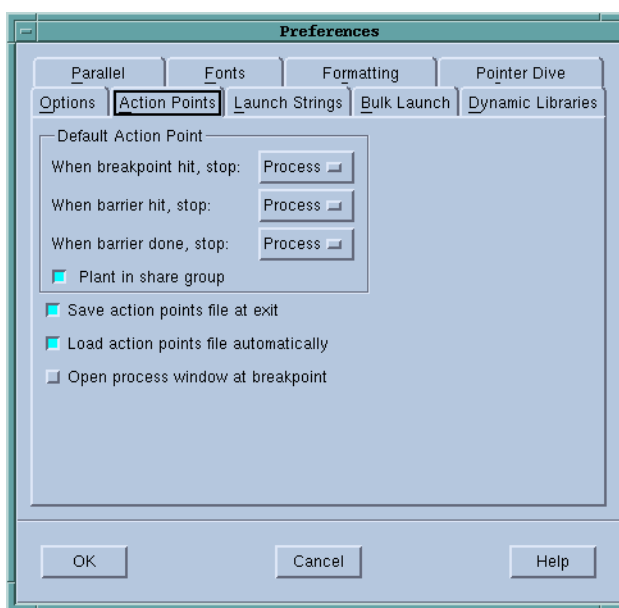
- "Action Points Page" on page 12

- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

Action Points Page

The commands within this page set the default values for the properties assigned when they are created. Some of these commands define what TotalView will do when it encounters an action point. Other commands tell TotalView that it should automatically save information about action point to a file so the action points can be reloaded at a later time. In this way, you do not have to reset action points every time you start TotalView. (See Figure 9)

Figure 9: Action Points Page



For additional information, see **Action Point > Properties**.

Default Action Point

The four controls in this area define the properties that TotalView assigns to action points when you create them.

When breakpoint hit, stop

Indicates what else is stopped when TotalView encounters a breakpoint. Your options are:

group: When one thread reaches the breakpoint, TotalView stops all processes in its program group.

process: Just stop the process that hit the breakpoint.

thread: Just stop the thread that hit the breakpoint.

For more information, see the **TV::stop_all** variable.

When barrier hit, stop

Indicates what else is stopped when TotalView encounters a barrier breakpoint. Your options are:

group: When one thread reaches the barrier, TotalView stops all processes in its program group.

process: Just stop the process that hit the barrier.

thread: Just stop the thread that hit the barrier.

For more information, see the **TV::barrier_stop_all** variable.

When barrier done, stop

Indicates what occurs when all threads or processes are stopped at the barrier breakpoint. Note that the set of threads and processes that are stopped at a barrier breakpoint is called the *satisfaction* set.

group: When a barrier is satisfied, TotalView stops all processes in the control group.

process: When a barrier is satisfied, TotalView stops the processes in the satisfaction set.

thread: Only the threads in the satisfaction set are stopped; other threads are not affected. For process barriers, there is no difference between **process** and **thread**.

In all cases, the satisfaction set is released when the barrier is satisfied.

For more information, see the

TV::barrier_stop_when_done variable.

Plant in share group

If set, enabling and disabling an action point alters it in all members of the share group. If this button is not selected, you must enable and disable the action point in each share group member individually.

For more information, see the **TV::share_action_point** variable.

Save action points file at exit

When set, TotalView automatically saves action points to an action points file when you exit. For more information, see the **Action Point > Save All** command.

For more information, see the

TV::auto_save_breakpoints variable.

Load action points file automatically

When set, TotalView automatically loads action points when it loads a file. For more information, see the **Action Point > Load All** command.

For more information, see the `TV::auto_load_actionpoints` variable.

Open process window at breakpoint

If selected, TotalView opens or raises the Process Window when your program reaches a breakpoint. Unlike other preferences on this page, this preference changes how existing action points behave.

For more information, see the `TV::pop_at_breakpoint` variable.

For information on other Preference pages, see:

- "Options Page" on page 9
- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

Launch Strings Page

You can set the launch strings for the following:

- "Single Debug Server Launch" on page 14
- "Visualizer Launch" on page 16
- "Source Code Editor" on page 17

Single Debug Server Launch

You can modify the TotalView Debugger Server (**tvdsvr**) Auto Launch feature by specifying the parameters TotalView will use to start this server on a remote host. By default, TotalView attempts to use the **rsh** command (**remsh** on HP-UX). Chapter 4 of the *TotalView User's Guide* contains a detailed description of these operations, along with instructions for starting the server manually if that becomes necessary.

Here is a brief summary of the automatic feature:

Enable single debug server launch

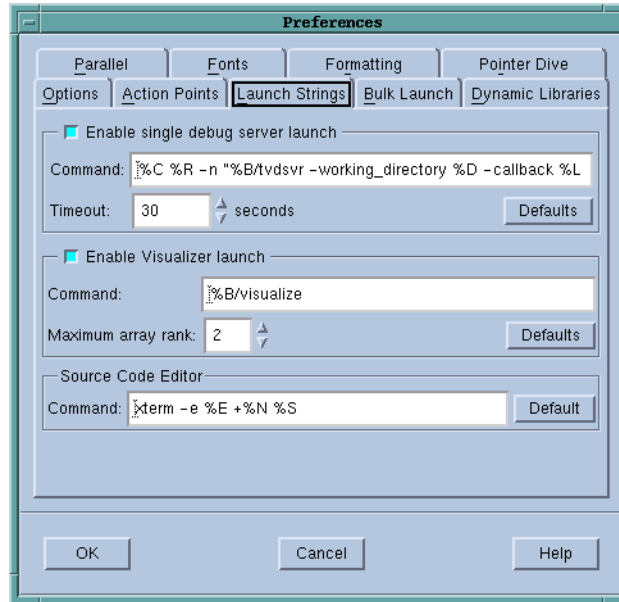
When this box is checked, TotalView automatically starts a server process when you ask it to debug a process on a remote host.

For more information, see the `TV::server_launch_enabled` variable.



*Even if you have enabled bulk server launch, you probably also want this option to be enabled. TotalView uses this launch string when you start TotalView upon a file when you have named a host within the **File > New Program** dialog box or have used the **-remote command line** option. You only want to disable single server launch when it can't work.*

Figure 10: Launch Strings Page



Command	The command TotalView uses when it starts the remote server. You must include the <code>-callback</code> and <code>-set_pw</code> arguments. For more information, see the <code>TV::server_launch_string</code> variable.
Timeout	Time in seconds that TotalView will wait before giving up trying to establish a connection. For more information, see the <code>TV::server_launch_timeout</code> variable.
Defaults	Changes the values defined within this area to their default values. This action overrides changes you have made using this preference, or to values set using options or X resources.

The expansion strings and options that you can use in the launch command string are:

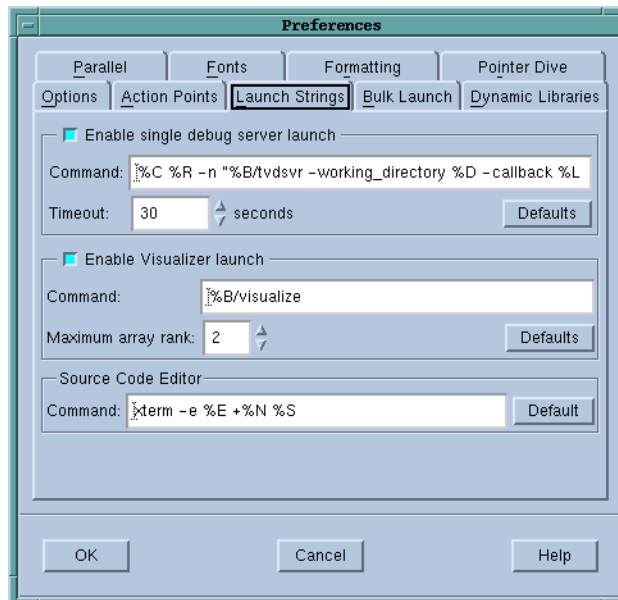
%C	Expands to the default name of the command used to start a remote process. If defined, the value of the environment variable <code>TVDSVRLAUNCHCMD</code> is used. Otherwise, the default name is <code>rsh</code> (<code>remsh</code> on HP-UX).
%R	Expands to the hostname of the remote machine as specified in the File > New Program dialog box.
-n	Tells the remote shell to read standard input from <code>/dev/null</code> .
-working_directory %D	Expands to the full path of the current working directory in which TotalView is running. The default command string tells <code>tvdsvr</code> to first try to <code>cd</code> to this directory. This directory name may be inappropriate if

	the target system's file system is not organized the same way as the host's file system.
-callback	Tells the server to call back to TotalView. This must be followed by the hostname and TCP/IP port number to call back to.
%L	Expands to the hostname and TCP/IP port number on which TotalView is listening for connections from tvdsvr .
%H	Expands to the hostname on which TotalView is running.
%S	Expands to the TCP/IP port number on which TotalView is listening for connections from tvdsvr .
-set_pw	Sets a 64-bit password for security. TotalView must supply this password when tvdsvr establishes the connection with it.
%P	Expands to the password that TotalView automatically generated.
%V	Expands to the TotalView verbosity setting. This launches the TotalView Debugger Server with the same verbosity level as TotalView.
%F	Contains the "tracer configuration flags" that need to be sent to tvdsvr processes. These are system-specific startup options that the tvdsvr process needs.

Visualizer Launch

The launch string defined within this area indicates how TotalView will launch a visualizer.

Figure 11: Launch Strings Page



Commands within this area are:

Enable Visualizer launch

When checked, TotalView will automatically attempt to start a visualizer process when it encounters a visualization command. If this is not checked, TotalView will not launch a visualizer even if you select the **Tools > Visualize** command or have used a **\$visualize** intrinsic.

For more information, see the **TV::visualizer_launch_enabled** variable.

Command

The command TotalView uses when it starts a visualizer. If you are using your own visualizer, you would enter its startup command here.

For more information, see the **TV::visualizer_launch_string** variable.

Maximum array rank

Sets the maximum rank. Edit this value if you plan to save the data exported from TotalView or display it in a different visualizer.

The maximum value you can enter is 16 and the default value is 2.

For more information, see the **TV::visualizer_max_rank** variable.

Defaults

Changes the values defined within this area to their default values. This action overrides changes you have made using this preference, or values set using command-line options.

Source Code Editor

The source code editor launch string area allows you to specify the command string TotalView will use to start an editor when you use the Process Window's **File > Edit Source** command. TotalView expands this string into a command that is executed by the **sh** shell.

Items recognized in the launch command string are:

%E Expands to the value of the **EDITOR** environment variable, or to **vi** if **EDITOR** is not set.

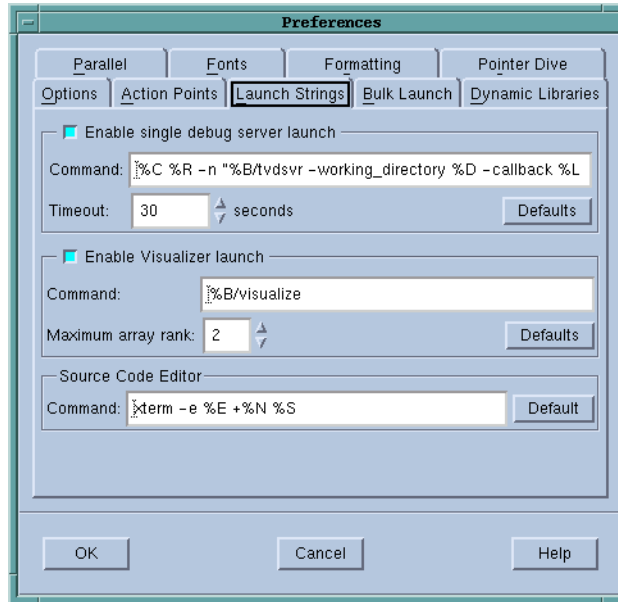
%N Expands to the line number in the Process Window's Source Pane.

%S Expands to the file name of the source file displayed in the Process Window's Source Pane.

%F Expands to the font name with which you started TotalView.

Default Changes the values defined within this area to their default values. This action overrides changes you have made using this preference, or values set using options or X Resources.

Figure 12: Launch Strings Page



The default editor launch string is `xterm -e %E +%N %S`.

For more information, see the `TV::editor_launch_string` variable. For information on other Preference pages, see:

- "Options Page" on page 9
- "Action Points Page" on page 12
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

Bulk Launch Page

The Bulk Launch Page allows you to specify the parameters TotalView will use when it does a bulk launch of TotalView Debugger Servers on remote hosts.

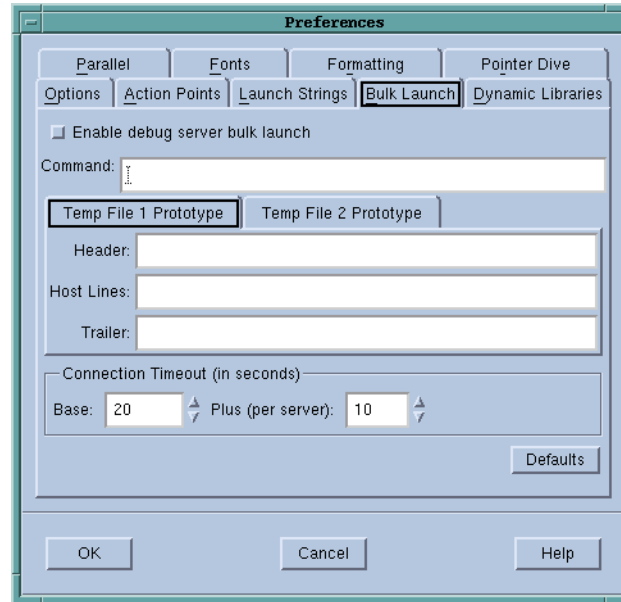
Information on bulk launching is contained within Chapter 4 of the TotalView Users Guide

Enable debug server bulk launch

When this box is checked, TotalView will start multiple server processes using a bulk launch command.

For more information, see the `TV::bulk_launch_enabled` variable.

Figure 13: Bulk Launch Page



When you enable bulk server launch, you probably do not want to disable single server launch. TotalView uses the single server launch string when you start TotalView upon a file when you have named a host within the **File > New Program** dialog box or have used the **–remote** command line option. You only want to disable single server launch when it can't work.

Command	The command TotalView uses when it attempts to start the (remote) servers. This information is stored within the TV::bulk_launch_string variable. Items that you can use in the bulk launch command string are:
%C	Expands to the default name of the command used to start a remote process. If defined, the value of the environment variable TVDSVRLAUNCHCMD is used. Otherwise, the default name is rsh (remsh on HP-UX).
%D	Expands to the full path of the directory to which TotalView is connected. The default command string tells tvdsvr to first try to cd to this directory. This directory name may be inappropriate if the target system's file system is not organized the same way as the host's file system.
%F	Contains the "tracer configuration flats" that need to be sent to tvdsvr processes. These are system-specific startup options that tvdsvr processes need.
%N	Expands to the number of servers that will be launched.
%H	Expands to the hostname on which TotalView is running.
%L	Expands to a comma-separated list of remote <i>host:port</i> for all remote hosts.

%R	Expands to a comma-separated list containing the entire remote host list.
%S	Expands to the comma-separated list containing the entire port list.
%P	Expands to the comma-separated list containing the entire password list.
%V	Expands to the TotalView verbosity setting. Setting this value allows the TotalView Debugger Server (tvdsvr) to be launched with the same verbosity level as TotalView.
%t1	Expands to the file name of temporary file number 1 (see Temporary Files later in this section).
%t2	Expands to the file name of temporary file number 2 (see Temporary Files later in this section).

Temp File Prototypes

Discussed later in this section.

This information is stored in variables beginning with **TV::bulk_launch_tmp**.

Connection timeout (in seconds)

Time to wait before giving up trying to establish the connections. The total timeout is calculated as a **Base** value, **Plus** an amount for each server launched.

For more information, see the **TV::bulk_launch_incr_timeout** variable.

Temporary Files

The bulk server launch facility allows you to create temporary files whose names are passed in the bulk server launch command. Each of these file has a *Header* line, followed by one line for each remote *Host*, followed by a *Trailer* line. Each tab within this page defines one set of these three files. The substitutions available in the header and trailer lines are those available in the bulk launch command just described.

Items that you can use in the host lines of a temporary file are:

%C	Expands to the default name of the command used to start a remote process. If defined, the value of the environment variable TVDSVRLAUNCHCMD is used. Otherwise, the default name is rsh (remsh on HP-UX).
%R	Expands to the hostname of the remote machine as specified in the File > New Program command.
-working_directory %D	Expands to the full path of the current working directory on which TotalView is running. The default command string tells tvdsvr to first try to cd to this directory. This directory name may be inappropriate if the target system's file system is not organized the same way as the host's file system.
-callback	Tells the server to call back to TotalView. This must be followed by the hostname and TCP/IP port number to call back to.

%L	Expands to the hostname and TCP/IP port number on which TotalView is listening for connections from tvdsrv .
%H	Expands to the hostname on which TotalView is running.
%S	Expands to the TCP/IP port number on which TotalView is listening for connections from tvdsrv .
-set_pw	Sets a 64-bit password for security. TotalView must supply this password when tvdsrv establishes the connection with it.
%P	Expands to the password that TotalView automatically generated.
-verbosity %V	Expands to the TotalView verbosity setting. This allows the TotalView Debugger Server to be launched with the same verbosity level as TotalView.

For more information, see "TotalView Command Syntax" in the *TotalView Reference Guide*.

For information on other Preference pages, see:

- "Options Page" on page 9
- "Action Points Page" on page 12
- "Launch Strings Page" on page 14
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

Dynamic Libraries Page

The controls within this page manage two different library behaviors:

- The top controls allow you to control if TotalView should stop execution when a named shared library is loaded. In most cases, you would do this so that you can set a breakpoint.
- The bottom controls tell TotalView how much information it should read when a shared library is loaded. (See "Symbol Loading" on page 22.)



*The **Default** button at the bottom of this page sets all fields on this page to their initial values.*

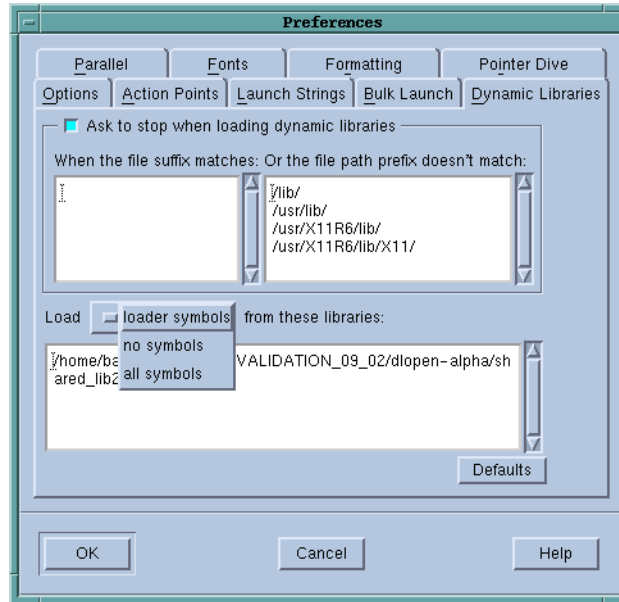
Stopping Before Executing

The controls in the top area tell TotalView if it should ask you if it is alright to load dynamic libraries and if it will, you can indicate which libraries it should ask questions about and which it should just load.

Ask to stop when loading dynamic libraries

If selected, TotalView uses the shared library path and file suffix to determine if it should ask if it should stop processes that load a shared library. The decision it makes is based on what you type in the two text areas.

Figure 14: Dynamic Libraries
Page



For more information, see the `TV::ask_on_dlopen` variable.

When the file suffix matches

Enter the suffixes that TotalView uses when it decides whether it will ask if it should stop the process when it loads a dynamic library. If the library being opened has a suffix that is on this list, TotalView asks if it should stop the process. Each suffix must reside on its own line. By default, this list is empty.

This list is global. It applies to all processes in this TotalView session.

For more information, see the `TV::dll_stop_prefix` variable.

And the file path prefix doesn't match

Enter prefixes that TotalView uses when it decides whether it will ask if it should stop the process when it loads a dynamic library. If the shared library being opened has a prefix that is on this list, TotalView does not ask if should stop the process. Each prefix must be on its own line. By default, this list is empty.

The list you specify here is global. It applies to all processes you examine in this TotalView session.

For more information, see the `TV::dll_ignore_prefix` variable.

Symbol Loading

The three items on the **Load from these libraries** list control whether TotalView reads loader and debugging symbols when it opens a library. Here's what placing entries into these areas means:

all symbols	TotalView reads all symbols. This is the default. Only enter a library name if it would be excluded by a wildcard in the loader symbols and no symbols areas. For more information, see the <code>TV::dll_read_all_symbols</code> variable.
loader symbols	TotalView only reads a library's loader symbols. If your program uses a number of large shared libraries that you will not be debugging, you might set this to <code>*</code> . You would then enter the names of DLLs that you need to debug in the all symbols area. For more information, see the <code>TV::dll_read_loader_symbols_only</code> variable.
no symbols	Normally, you wouldn't name any libraries on this list as TotalView may not be able to create a backtrace through this library if it doesn't have these symbols. However, you can sometimes increase performance if you place the names of your largest libraries here. For more information, see the <code>TV::dll_read_no_symbols</code> variable.

When reading a library, TotalView looks at these lists in the following order:

- 1 all symbols**
- 2 loader symbols**
- 3 no symbols**

That is, TotalView processes these lists in order. This means that if you name a library in more than one list, TotalView ignores the second (or third) references to the library.

When entering library names, you can use the `*` and `?` wildcard characters. For example:

<code>*mystuff*</code>	Matches <code>./lib/libmystuff.so</code> as well as anything else that contains the mystuff string in its filename.
<code>/home/myname/dev/*</code>	Matches any library in the <code>/home/myname/dev</code> directory.
<code>*</code>	Matches every library that TotalView would read.

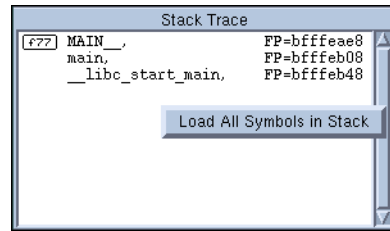
If your program stops in a library that has not had its symbols read, TotalView reads its symbols before reporting the error that caused execution to stop.

You can tell TotalView that it should automatically read a library's symbols when it stops by setting the `TV::auto_read_symbol_at_stop` variable.

Load All Symbols in Stack Context Menu Command

If you place the cursor in the Stack Trace Pane and click your right mouse button, TotalView displays the **Load All Symbols in Stack** command. Selecting this command tells TotalView to examine the stack trace for the current thread and finds any frames where the thread was executing in a library that has not had all its symbols read. If TotalView locates any libraries, it reads in their debugging symbols.

Figure 15: Load All Symbols in Stack Context Menu



If, while reading in these libraries, it discovers other libraries that must be read in, it will also read in these additional symbols.

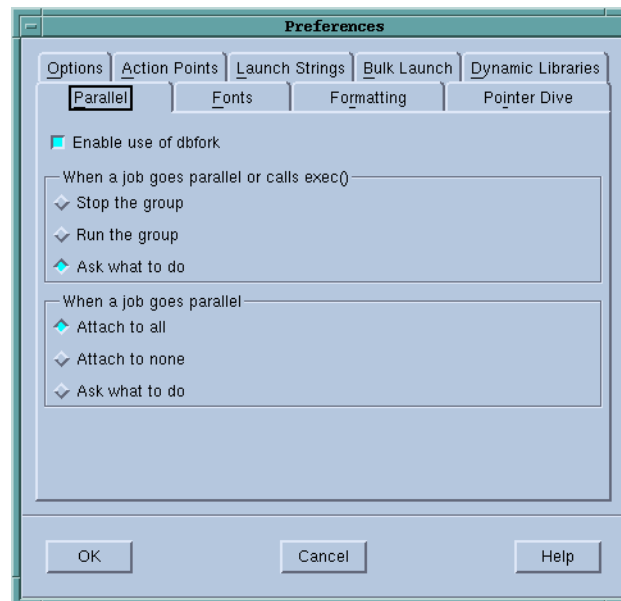
For information on other Preference pages, see:

- "Options Page" on page 9
- "Action Points Page" on page 12
- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

Parallel Page

The preferences on this page tell TotalView that it should **dbfork**. It also indicates what will happen when your program goes parallel.

Figure 16: Parallel Page



Enable use of dbfork

When set, TotalView catches the **fork()**, **vfork()**, and **execve()** system calls if your executable is linked with

the **dbfork** library. This is discussed in Chapter 8 of the *TotalView Reference Guide*.

For more information, see the **TV::dbfork** variable.

When a job goes **parallel** or calls **exec()**

The buttons in this area have the following meaning:

Stop the group

Stop the control group immediately after the processes are created.

Run the group

Allows all newly created processes in the control group to run freely.

Ask what to do

If set, TotalView asks if it should start the created processes.

For more information, see the **TV::parallel_stop** variable.

When a job goes **parallel**

The buttons in this area have the following meaning:

Attach to all

TotalView automatically attaches to all processes when they begin executing.

Attach to none

TotalView will not attach to any created process when it begins executing.

Ask what to do

If set, TotalView opens the same dialog box that it displays when you select **Group > Attach Subset** command. Using this dialog box, you tell TotalView to which processes it should attach. Note that TotalView does not display this dialog box when you set the preference. Instead, this preference tells TotalView that it should display the dialog box when it is about to create processes.

For more information, see the **TV::parallel_attach** variable.

For information on other Preference pages, see:

- "Options Page" on page 9
- "Action Points Page" on page 12
- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Fonts Page" on page 26
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

Fonts Page

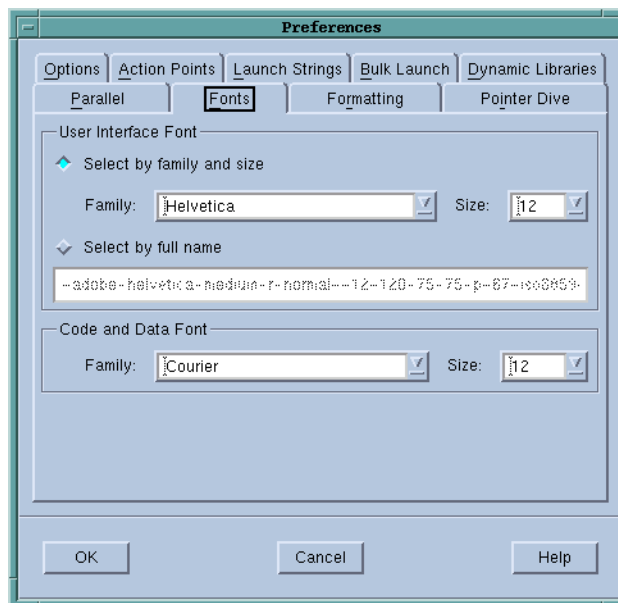
TotalView uses two fonts, a fixed width and a variable width font. Program data is displayed in a fixed width font. User Interface menus, buttons, labels, and dialog boxes use a variable width font.

The fonts you can select are those already installed with your X server.

You can select a variable width font by either selecting the font family and size or by entering the exact font name. In the first case, TotalView will attempt to select a compatible font. In the second, TotalView uses the name you selected.

The following controls set the user interface font. This is the font TotalView uses when it wants to display information using a variable width font. For the most part, this is the information that is not part of your code. (See Figure 17 on page 26.)

Figure 17: Fonts Page



Select by family and size

Use the controls in this area to indicate the **Family** and the **Size** of the variable width font. The font **Family** indicates the kind of font that will be used; for example, **Helvetica** or **Times Roman**. The **Size** indicates the point size at which TotalView displays characters in the **Family** are displayed.

Select by full name

When you select a font name, you must supply the complete font name. The **xlsfonts** program supplied with your X server lists the fonts you can use.

The remaining settings are used when TotalView displays code and data. Using these controls, you can also select a font family and a font size.

For information on other Preference pages, see:

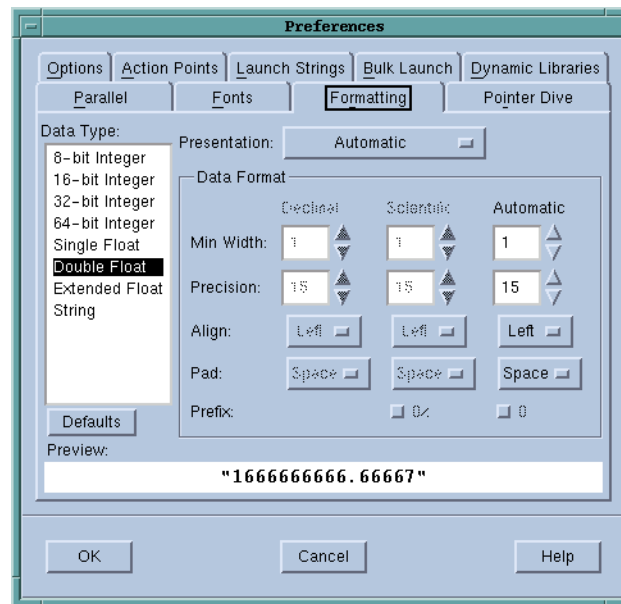
- "Options Page" on page 9
- "Action Points Page" on page 12
- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Formatting Page" on page 27
- "Pointer Dive Page" on page 29

Formatting Page

The controls within this page specify the precision at which TotalView should display a variable's value. You can define the precision for the following data types:

- 8-bit integers
- 16-bit integers
- 64-bit integers
- Single precision floating point numbers
- Double precision floating point numbers
- Extended precision floating point numbers
- Strings

Figure 18: Formatting Page



The variables set by this preference begin with `TV::data_format`.

If you have selected a numeric data type in the left hand list, the presentations you can use are:

Selection	Tells TotalView To Display Values:
Automatic	TotalView to display information as hex(dec) in C and C++ and dec(hex) in Fortran.
Hexadecimal (Decimal)	In both hexadecimal and decimal. The decimal value is displayed within parentheses.
Decimal (Hexadecimal)	In both decimal and hexadecimal. The hexadecimal value is displayed within parentheses.
Decimal	As decimal numbers.
Hexadecimal	As hexadecimal numbers.
Maximum Length	If the data type selected is String, the display changes to a single box and up and down controls that let you specify the maximum number of characters that will be in the displayed string.
Octal	As octal numbers.
Scientific	Using scientific notation.

After you have selected a data type, you can specify the precision using the following controls:

Format	Meaning
Min Width	The total number of positions used to display a number. This value includes decimal points as well as any other characters contained within the display. If this width is too small to display a value, TotalView will use more characters.
Precision	The number of characters to the right of the decimal point.
Align	Selecting Left or Right tells TotalView how it should display information within the specified width. For example, if the Min Width is 20 and TotalView needs only 12 characters to display a value, the value can be placed to the right with 8 preceding spaces or to the left with 8 trailing spaces.
Pad	If the Align value is Right and TotalView needs fewer positions to display the value than indicated in the Min Width control, it can print the leading spaces as either Spaces or Zeroes .
Prefix	If TotalView is displaying Hexadecimal numbers, it can include the 0x hexadecimal indicator with the number. Similarly, when it is displaying Octal numbers, it can include the 0 octal indicator.

As you change values, the **Preview** area shows the effect of your changes.



Before making changes, it may be helpful to set a large minimum width, then play with the other controls to see what happens as you alter how TotalView will display values.

For information on other Preference pages, see:

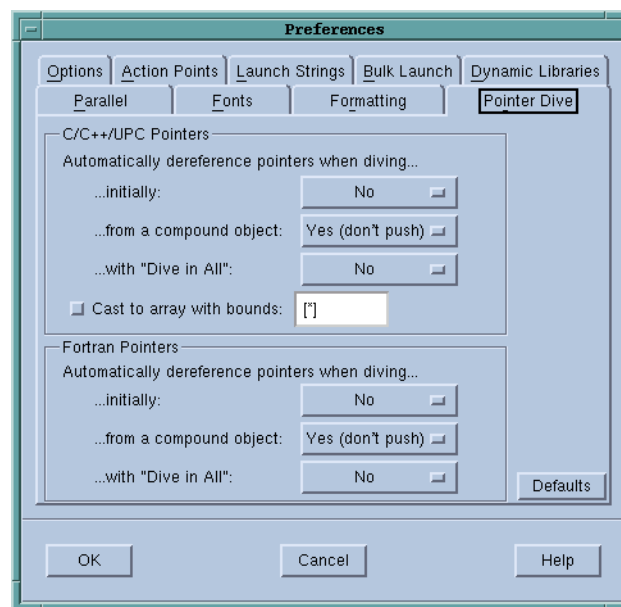
- "Options Page" on page 9
- "Action Points Page" on page 12

- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Pointer Dive Page" on page 29

Pointer Dive Page

The controls within this page tell TotalView if it should automatically dereference pointers when diving. One set of controls is used for C, C++, and UPC. The other set of controls is for Fortran. (See Figure 19 on page 29.)

Figure 19: Pointer Dive Page



The shared controls let you automatically dereference pointers when diving as follows:

- ...initially** Tells TotalView what it should do when you dive on a variable.
- ...from a compound object** Tells TotalView what it should do when you dive on an element in a compound data element such as a structure.
- ...with "Dive in All"** Tells TotalView what it should do when you dive on a variable using the **Dive in All** command.

You can specify one of the following for each of these controls:

- No** Do not automatically dereference variables when diving.
- Yes** Automatically dereference variables. In addition, place the variable on the "variable" stack so that you can use

the **View > Undive** command to see the pointer's value.

Yes (don't push)

Automatically dereference variables. Do not place the variable on the "variable" stack. This means that you cannot use the **View > Undive** command to see the pointer's value.

The **Cast to array with bounds** control tells TotalView that it should assume that the pointer to is pointing to an array of values when it dereferences a C or C++ pointer. The text box lets you state how many items exist in the array.

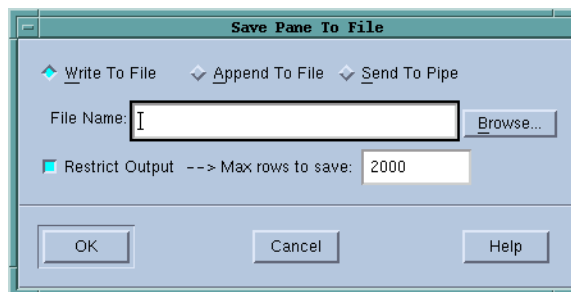
For information on other Preference pages, see:

- "Options Page" on page 9
- "Action Points Page" on page 12
- "Launch Strings Page" on page 14
- "Bulk Launch Page" on page 18
- "Dynamic Libraries Page" on page 21
- "Parallel Page" on page 24
- "Fonts Page" on page 26
- "Formatting Page" on page 27

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

Figure 20: File > Save Pane Dialog Box



Write to File

Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File

Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist,

TotalView creates the file before writing this information.

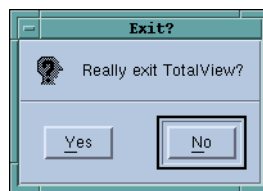
Send To Pipe Sends the data to the program or script named in the **File Name** field.

Restrict Output --> Max rows to save
If checked, TotalView will limit how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

Figure 21: File > Exit Dialog Box



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 31
- **Edit > Cut** on page 31
- **Edit > Copy** on page 32
- **Edit > Paste** on page 32
- **Edit > Delete** on page 32
- **Edit > Find** on page 32
- **Edit > Find Again** on page 33

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language state-

ment contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

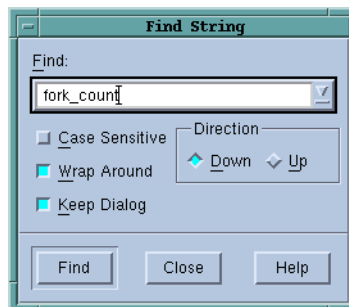
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 22: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for |

	"foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box.
<i>Direction</i>	Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the Find box.
Close	Closes the Find dialog box.

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes a search defined by the **Find** command. TotalView begins searching at the current text cursor position. The direction in which TotalView searches is the same as the last search you defined.

View Menu Commands

The commands on the **View** pulldown are:

- **View > Dive** on page 33
- **View > Dive in New Window** on page 33
- **View > Expand All** on page 34
- **View > Collapse All** on page 34
- **View > Display Manager Threads** on page 34
- **View > Display Exited Threads** on page 34

View > Dive

Opens the selected process or thread in a Process Window. If the process already owns a Process Window, that window is moved to the front of the display. If a Process Window does not exist, TotalView creates a window for it or replaces the contents in an existing window. Note, that TotalView tries very hard to reuse Process Window.

If a Process Window already exists, the window is focused on an arbitrary thread. If the process has no open windows, TotalView opens a window containing the first thread in the process.

If you want the dive operation to create a new window, use the **View > Dive in New Window** command

View > Dive in New Window

Opens the selected process or thread in a Process Window. Unlike a **View > Dive in New Window** command, this will create a window for a new process.

If a Process Window already exists, the window is focused on an arbitrary thread. If the process has no open windows, TotalView opens a window containing the first thread in the process.

View > Expand All

Expands all trees that are not displaying all of its information. That is, this is equivalent to selecting every + icon within the Root Window.

View > Collapse All

Collapses all trees. That is, this is the equivalent to selecting every – icon within the Root Window.

View > Display Manager Threads

When selected (which is the default), TotalView displays manager threads. Manager threads are threads created by the operating system that support your program's activities. In most cases, you are not interested in these threads, so unchecking this command is usually what you want to do.

View > Display Exited Threads

When selected (which is the default), TotalView displays exited threads. When debugging a multithreaded application, tracking threads as your program creates and deletes them can be difficult. When this command is selected, TotalView doesn't remove them from the display.

Tools Menu Commands

The commands on the **Tools** pulldown are:

- **Tools > Restart Checkpoint** on page 34
- **Tools > P/T Set Browser** on page 36
- **Tools > PVM Tasks** on page 37
- **Tools > Command Line** on page 37

Tools > Restart Checkpoint

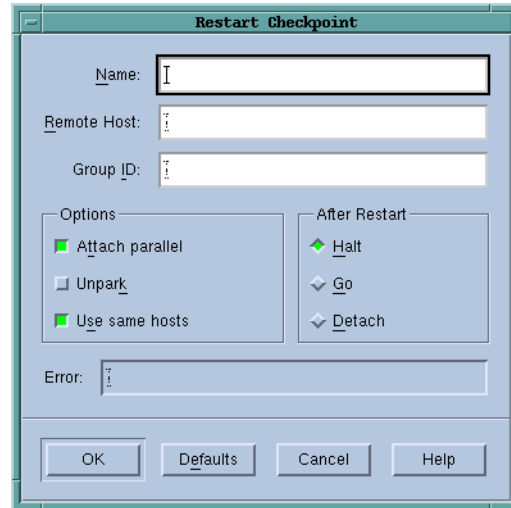
Use this dialog box to restore and restart all of the checkpointed processes. By default, TotalView attaches to the base process. If parallel processes are related to this base process, TotalView attaches to them. If you do not want TotalView to automatically attach to them, deselect the **Attach parallel** option.

If an error occurs while attempting to restart the program from the checkpoint, information is displayed in the **Error** area.

The CLI's **drestart** command performs the same operations as this command.

Name	Names a previously saved checkpoint file.
Remote Host	Names the remote host upon which the restart will occur.
Group ID	Names the control group into which TotalView places all created processes
Options	Indicates control options that you may find useful. If this is an RS/6000 checkpoint, Attach parallel is auto-

Figure 23: Tools > Restart Checkpoint



atically checked and it cannot be unchecked. These options have the following meaning:

Attach parallel

If selected, TotalView attaches to parallel processes as they are being created. If this item is not selected, TotalView only attaches to the base process.

Unpark

(SGI only) Select this checkbox if the checkpoint was created outside of TotalView or if you did not select the **Park** checkbox within the **Tools > Restart Checkpoint** dialog box when you created the checkpoint file.

Use Same Hosts

(IBM only) If selected, the restart operation tries to use the same hosts as were used when the checkpoint was created. If TotalView cannot use the same hosts, the checkpoint operation fails.

After Restart

Defines the state of the process both before and after the checkpoint. You can use one of the following options:

Halt

Parallel processes are held immediately after the place where the checkpoint occurred. TotalView attaches to these created parallel processes. (This is the default.)

Go

(SGI only) Checkpointed parallel processes are started and TotalView attaches to the created processes.

Detach

(SGI only) Checkpointed process are started. TotalView does not attempt to attach to them.

Restarting on AIX using LoadLeveler: On the RS/6000, if you wish to debug a **LoadLever poe** job from the point at which the checkpoint was made, you must resubmit the program as a **LoadLeveler** job to restart the checkpoint. You will also need to set the **MP_POE_RESTART_SLEEP** environ-

ment variable to an appropriate number of seconds. After you restart **poe**, start TotalView and attach to **poe**.



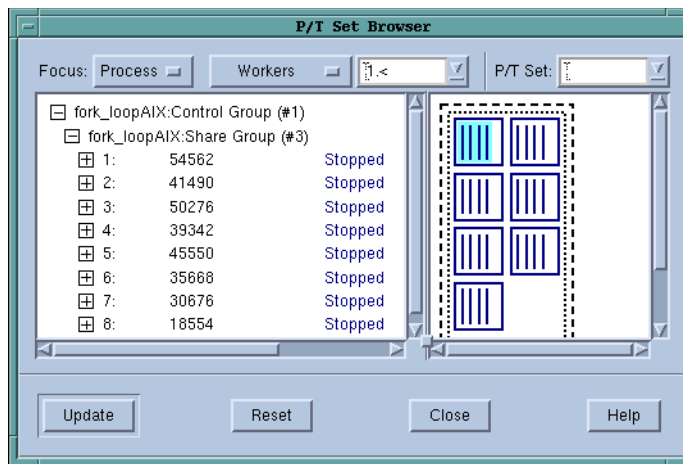
*When attaching to **poe**, parallel tasks will not yet be created, so do not try to attach to any of them. Also, you'll need to set the **Attach to none** option with the **Parallel** Page of the **File > Preferences** Dialog Box.*

When doing this, you cannot use the restart the checkpoint using this command. **poe** will tell TotalView when it is time to attach to the parallel task so that it can complete the restart.

Tools > P/T Set Browser

The P/T Set Browser shows the state of your program's threads and processes and visually shows the meaning of a P/T set. For example, it highlight all threads associated with the current share group. The controls at the top of the page let you specify a P/T set. The two panes in the middle let you individually select threads and processes as well as control and share groups.

Figure 24: Tools > P/T Set Browser Window



Use the following controls to select processes and threads:



You'll find extensive information on the meaning of these controls in the TotalView Users Guide.

Width

Limits TotalView's selection of what executing elements may be selected. Your choices are **All**, **Group**, **Process**, and **Thread**. This control tells TotalView what the group, process, or thread of interest.

Group

Limits TotalView's selection of what executing elements within the *Width* it should chose. Your choices are **Control**, **Share**, **Workers**, and **Lockstep**. For example, if you have chosen process width and select the lockstep group, you are telling TotalView that it should select all members of the lockstep group within the current process.

P/T Selector	Tells TotalView which thread should be displayed or which thread is the thread of interest. If you need to specify more than one thread, use the P/T Set control.
P/T Set	Allows you directly enter CLI commands that create a process or thread set. For example, you could create a union of P/T sets in this control.

You can individually select threads and processes within the two panes. Selecting an element on one side causes the corresponding element to be selected within the other. You can also select boxes that surround the processes and threads on the right hand side.

The numbers used to identify processes, threads, and groups in this window are the same as that used in the Root Window and in some areas of the Process Window.

Tools > PVM Tasks

See Chapter 8, "PVM Tasks Window," on page 169 for information.

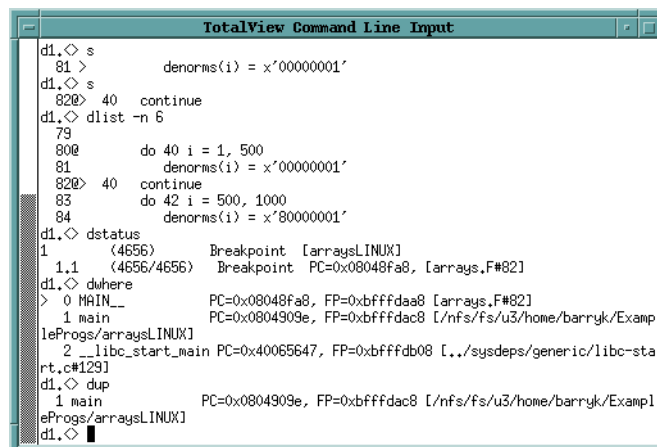
Tools > Memory Debugging

For information, see **Memory Debugging Window** on page 175.

Tools > Command Line

Opens the CLI window. This window is an **xterm** window into which you enter CLI commands.

Figure 25: Tools > Command Line (CLI) Window



```

TotalView Command Line Input
d1, < s
81 >          denorms(i) = x'00000001'
d1, < s
82@> 40  continue
d1, < dlist -n 6
79
80@      do 40 i = 1, 500
81          denorms(i) = x'00000001'
82@> 40  continue
83      do 42 i = 500, 1000
84          denorms(i) = x'80000001'
d1, < dstatus
1 (4656)      Breakpoint [arraysLINUX]
1.1 (4656/4656) Breakpoint PC=0x08048fa8, [arrays.F#82]
d1, < dwhere
> 0 MAIN__   PC=0x08048fa8, FP=0xbffffdaa8 [arrays.F#82]
1 main      PC=0x0804909e, FP=0xbffffdac8 [/nfs/fs/u3/home/barryk/Examp
leProgs/arraysLINUX]
2 __libc_start_main PC=0x40065647, FP=0xbffffdb08 [../sysdeps/generic/libc-sta
rt.c#129]
d1, < dup
1 main      PC=0x0804909e, FP=0xbffffdac8 [/nfs/fs/u3/home/barryk/Examp
leProgs/arraysLINUX]
d1, <

```

Information on using the CLI is located in the *TotalView Users Guide*. This book can also be accessed by using TotalView's **Help** command.

Window Menu Commands

The commands on the **Window** pulldown are:

- **Window > Update** on page 38
- **Window > Update All** on page 38
- **Window > Memorize** on page 38
- **Window > Memorize all** on page 38

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

The Process Window

2

Process Window Panes

The Process Window contains general information about the state of the process, with a summary of its current threads and their states. It also displays detailed information for one of the process's threads. This thread is called the "current thread."

The Process Window is divided into five panes:

- **Stack Trace Pane**, which displays the call stack.
- **Stack Frame Pane**, which displays information about the current thread's variables.
- **Source Pane**, which contains the source code or assembly instructions for your program.
- **Threads Pane**, which contains a list of all active threads in the process.
- **Action Points Pane**, which displays a list of the thread's current action points.

Immediately above these panes are several lines that display information about the process and thread being displayed:

- The *process bar* displays process status information. This information includes the process ID (PID), a process name, and a status indicator. If the process is running on a remote machine, this information is also displayed.
- The *thread bar* displays a thread ID, which is a combination of the PID and TID (thread ID) generated by TotalView and a status indicator. If the process is running on a remote machine, this information is also displayed.

Stack Trace Pane

Shows the call stack of routines that the selected thread is executing. You can move up and down the call stack by selecting the routine. When you select a different stack frame, TotalView updates the Stack Frame and Source Code *panes* to show the information about the routine that you just selected.

The information in this pane is as follows:

- The beginning of each line indicates the language in which that routine is written.
- The second column is the name of the routine.
- The final column indicates the location of the routine's frame pointer.

Stack Frame Pane

Displays all the function parameters, local variables, and registers for the selected stack frame. This frame does not include information on your program's global variables; use the **Tools > Program Browser** command to obtain this information.

This frame can be set in two ways:

- It is selected implicitly when TotalView hits a breakpoint or when TotalView loads a program.
- You select a routine in the Stack Frame Pane.

To change the value of any item in this pane, just click on the value you wish to change and then edit its value. To see more information about a variable or to dereference a chain of pointer variables, double-click the line containing the variable.

If you are debugging OpenMP code and the current thread is a slave thread in a parallel region, TotalView shows a special stack frame in the Stack Frame Pane.

Source Pane

Contains the source for the routine associated with the selected stack frame. The arrow in the left margin of the Source Pane indicates the location of the PC for that stack frame.

To set breakpoints in the process, click on the line number. A "stop sign" icon appears under the cursor. To clear a breakpoint, place your mouse over the stop sign for the breakpoint and click on it again. To set or alter an action point's settings, select the line and then select the **Action Point > Properties** command. (Right-clicking the line brings up a popup menu. You can select **Properties** from this menu.) Note that breakpoints apply to all threads in the process.

To view the source for a function or the contents for a variable whose name appears in the Source Pane, double-click on it or, after selecting it, use the **View > Dive** command. If you click on a function, TotalView shows the function in the source pane by replacing the information that was being shown. You can return the display to how it was previously by selecting the < indicator located to the right of the source pane.

If you click on a variable, information for the variable appears in a separate window.

Threads Pane

The *Threads Pane* shows the threads that currently exist in this process. The number in the Threads Pane's title is the number of threads that currently exist.

When you change to a different thread by selecting it in this list, TotalView updates other panes to show the information for that thread.

This pane has three columns, as follows:

- The first contains the thread ID, a slash, and the system ID.
- The second is the thread's status, as follows:

Character and Meaning	Definition
Bnn (Breakpoint)	Stopped at a breakpoint. <i>nn</i> is the ID of the breakpoint if it is a thread.
E (Error)	The Error state usually indicates that your program received a fatal signal from the operating system. Signals such as SIGSEGV , SIGBUS , and SIGFPE can indicate an error in your program.
H (Held)	Either you or TotalView is holding the thread. <i>Holding</i> means that the process or the thread cannot run until it is released. You can explicitly release it or TotalView will release it when the condition that caused it to be held is satisfied.
K (Kernel)	The thread is executing inside the kernel (that is, something made a system call). When a thread is in the kernel, the operating system does not allow TotalView to view the full state of the thread.
R (Running)	The thread is running or can run.
T (Stopped)	Stopped; however, the thread is not stopped at a breakpoint and because of an error.
W (Watchpoint)	Stopped at a watchpoint.

- The third column names the routine containing the PC.

Action Points Pane

The Action Points Pane shows the list of breakpoints, evaluation points, and watchpoints for the process. This pane has three columns, as follows:

- The first indicates the kind of breakpoint. Here you will see an icon indicating if you are at a breakpoint, evaluation point, barrier point, or watchpoint. TotalView displays the icon in gray if you had disabled or suppressed the action point. See **Action Point > Enable** on page 96 and **Action Point > Suppress All** on page 102 for more information.
- A TotalView action point identifier. These identifiers are never reused within a session. This identifier is more often used within the CLI than within the TotalView GUI.
- Text indicating where the breakpoint resides. This information includes a line number within a file, a program name, and the offset at which the breakpoint is set.

TotalView orders this list so that breakpoints are sorted by module name, routine name, line number, and address.

File Menu Commands

The commands on the **File** pulldown are:

- **File > New Program** on page 44
- **File > Search Path** on page 46
- **File > Signals** on page 47
- **File > Preferences** on page 48
- **File > Open Source** on page 48
- **File > Edit Source** on page 49
- **File > Save Pane** on page 49
- **File > Rescan Libraries** on page 50
- **File > Close Relatives** on page 50
- **File > Close** on page 50
- **File > Exit** on page 50

File > New Program

Use this dialog box to specify the name of a new executable file, to attach to an existing running process or core file, or to specify the location of the process.

The **New Program** dialog box allows you to load another program. When loading a program, you need to enter and indicate:

- The name of your program's executable file.
- Whether or not to attach to an existing process or core file.
- The location of the process, which can be **Local**, **Remote Host**, or **Serial Line**.

The simplest case is when you want to debug a new program on a local host. Type the name of a program you wish to debug in the **Executable** field and press **OK**.

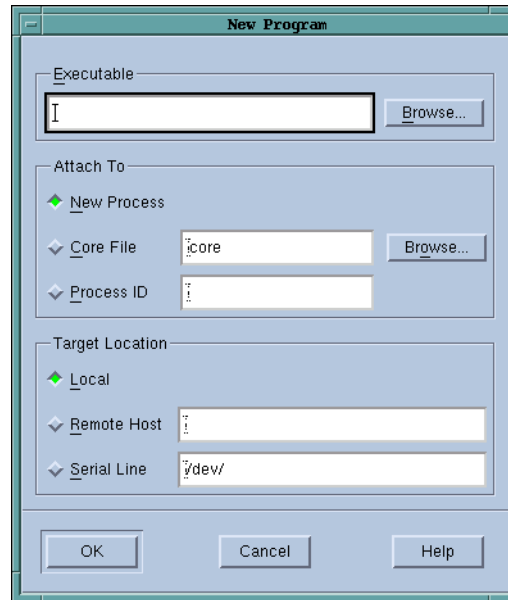
If you want to attach to an existing process or read a core file, specify a **Process ID** or enter a name in the **Core File** field. In both of these cases, you must also enter a pathname in the **Executable** field.

If you want to debug a process on a remote machine, enter the host name or IP address of the remote machine in the **Remote Host** field. (See Figure 26 on page 45.)

The fields in this dialog box and their meaning are:

Executable	The name of the executable file to be debugged. You can enter either a full or relative path name. If you enter just a file name, TotalView searches for the file in the directories you specified with the File > Search Path command and in all the directories named in your PATH environment variable. You can use the Browse button to search the file system for the file.
Attach To	Lets you attach to an already running program or to load a core file.

Figure 26: File > New Program Dialog Box



New Process

If selected, TotalView loads the executable. If the executable is already loaded, TotalView loads it again.

Core file

If selected, TotalView loads the core file. You must enter a program name in the **Executable** field because TotalView cannot know if this program is actually associated with the process.

You can use the **Browse** button to search the file system for the core file.

Process ID

If selected, TotalView loads the program associated with this process ID. A program name must be entered in the **Executable** field because TotalView cannot check that this program is actually associated with the process.

If this process is already loaded, TotalView raises the window; that is, it makes the process's window completely visible.

If the process has children that called **execve()**, TotalView tries to determine each child's executable. If TotalView cannot determine the executables for the children, you need to delete (kill) the parent process and start it under TotalView control.

If the executable is a multiprocess program, TotalView asks if you want to attach to all relatives

of the process. To examine all processes, select **Yes**.



*This is the default behavior. You can change this behavior by using commands within the **File > Preference's Parallel Page**.*

Target Location Lets you indicate the program's location, as follows:

Local The program is on your current machine.

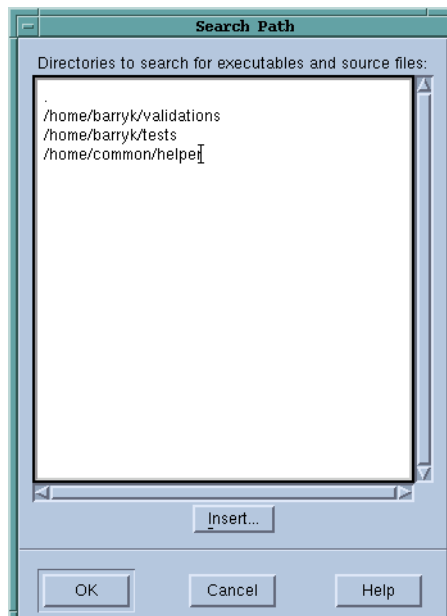
Remote Host The program is on a different machine and TotalView will access it over your network.

Serial Line The program is on a different machine and TotalView will access it over a serial line.

File > Search Path

Use this dialog box to set the directories in which TotalView will search for executable and source files. You can type a directory name within the text edit box and you can use the **Insert** button to graphically move through your system's file system to select a directory to be inserted.

Figure 27: File > Search Path Dialog Box



TotalView searches for source files, in the following order:

- 1 The current working directory (.).
- 2 The directories you specify by using the **File > Search Path** command in the exact order you enter them.
- 3 If you entered a full path name for the executable when you started TotalView, TotalView searches this directory.

- 4 If your executable is a symbolic link, TotalView will look in the directory in which your executable actually resides for the new file.

As you can have multiple levels of symbolic links, TotalView keeps on following links until it finds the actual file. After it has found the current executable, it will look in its directory for your file. If it isn't there, it'll back up the chain of links until either it finds the file or determines that the file can't be found.

- 5 The directories specified in your **PATH** environment variable.



The search path is local to the machine upon which TotalView is running. TotalView does not search for files on remote hosts.

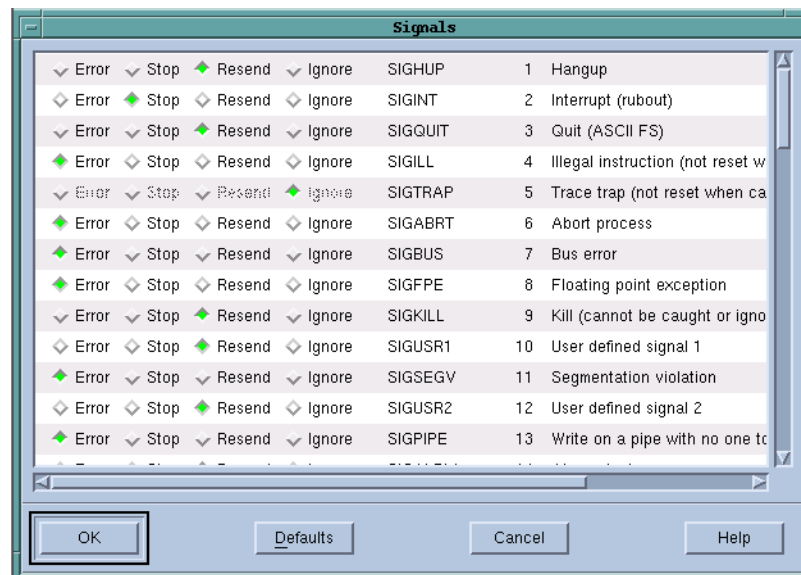
File > Signals

Use this dialog box to control the way TotalView responds to UNIX signals. Here are some special considerations:

- In most cases, you should not select **Ignore** for signals such as **SIGSEGV** or **SIGBUS** that indicate that an error occurred. Modifying the behavior of these signals is unlikely to do what you want and may cause TotalView to get caught in a fault loop with your program.
- You cannot alter **SIGTRAP** and **SIGSTOP** because these signals are used internally by TotalView.

If several processes encounter errors simultaneously, TotalView only opens a window for the *first* error. Thus, if 64 processes in a parallel program try to divide by zero at the same time, TotalView will not open 64 process windows simultaneously; instead, it will only raise one window.

Figure 28: File > Signals Dialog Box



The buttons indicate what TotalView should do when a signal is raised. The actions TotalView can perform are:

Error	Stop a process, place it in the error state, and display an error in the title bar of the Process Window. If the Stop control group on error check box within the Preference's Option Page is selected, TotalView also stops all related processes. Select this mode for severe error conditions such as SIGSEGV and SIGBUS signals.
Stop	Stop a process and place it in the stopped state; that is, stop the process and take no further action. Its status will be shown as T in the Root Window. Select this mode if you want TotalView to handle this signal as if it were a SIGSTOP signal.
Resend	Immediately forward the signal to the process. From your program's point of view, the only difference between TotalView handling this signal and how it is handled otherwise is that your process receives the signal a little slower than it normally would. By default, the common signals for terminating a process (SIGKILL and SIGHUP) use this mode.
Ignore	Discard the signal and continue the process as if it had not occurred.

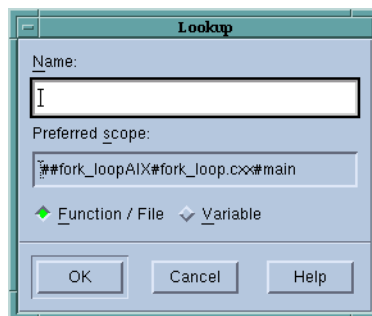
File > Preferences

Use this dialog box to set preferences for how TotalView will behave situations, as well as define some general characteristics. For more information, see "File > Preferences" on page 9, which is within the Root Windows help.

File > Open Source

After entering the name of one of your program's source files, TotalView will display this file within its Source Pane. If a header file contains executable code, you can enter a header file name.

Figure 29: File > Open Source Dialog Box



Notice that this is the same dialog box that TotalView displays when you select the **View > Lookup Function** command.

File > Edit Source

Tells TotalView to open the file associated with the contents of the Source Pane in a text editor.

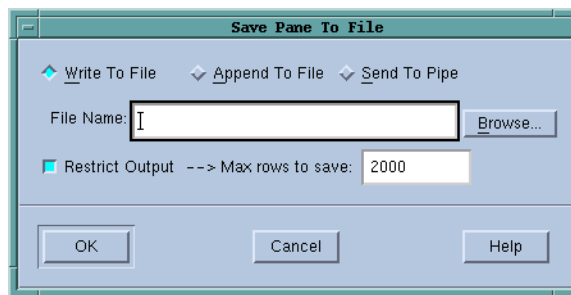
TotalView uses an editor launch string to determine how to start your editor. TotalView expands the editor launch string into a command that is then executed by the **sh** shell.

You can set the command that TotalView uses when it launches a text editor by setting the **Source Code Editor** launch string. For more information, see **File > Preferences** on page 48.

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

Figure 30: File > Save Pane Dialog Box

**Write to File**

Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File

Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist, TotalView creates the file before writing this information.

Send To Pipe

Sends the data to the program or script named in the **File Name** field.

Restrict Output --> Max rows to save

If checked, TotalView will limit how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

File > Rescan Libraries

Scans the shared library (dynamic link) information looking for new information. If new information is found, it is reloaded. If, however, the information is up-to-date, no updating occurs.

You would use this command when you have recompiled or moved a shared library. This command was sometimes needed in previous versions of TotalView. It is seldom, if ever, needed currently. However, it does exist if an unforeseen problem occurs.

File > Close Relatives

Closes windows that were created using controls on this window and also closes similar windows. If any of these windows had also created windows (for example, creating a Variable Window from a Image List Window), TotalView also closes these secondary windows.

While TotalView does not close the current Process Window, Process Windows related to this window are closed.

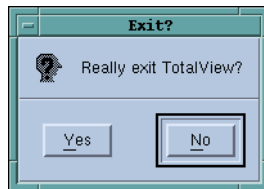
File > Close

Closes the Process Window. TotalView does not close other windows that were spawned from it. For example, if you had created one or more Variable Windows from within a Process Window, TotalView does not close these windows.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

Figure 31: File > Exit Dialog Box



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The following commands are on the **Edit** pulldown:

- **Edit > Undo** on page 51
- **Edit > Cut** on page 51
- **Edit > Copy** on page 51
- **Edit > Paste** on page 51
- **Edit > Delete** on page 51
- **Edit > Find** on page 52
- **Edit > Find Again** on page 52

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 32: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. By selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option. |
| Keep Dialog | If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box. |
| <i>Direction</i> | Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file." |
| Find | Tells TotalView to search for the text within the Find box. |
| Close | Closes the Find dialog box. |

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The following commands are on the **View** pulldown:

- **View > Dive** on page 53
- **View > Dive in New Window** on page 54
- **View > Undive** on page 54
- **View > Redive** on page 54
- **View > Reset** on page 54
- **View > Lookup Function** on page 55
- **View > Lookup Variable** on page 55
- **View > Next Process** on page 57
- **View > Previous Process** on page 57
- **View > Next Thread** on page 57
- **View > Previous Thread** on page 57
- **View > Source As > Source** on page 57
- **View > Source As > Assembler** on page 57
- **View > Source As > Both** on page 57
- **View > Assembler > Symbolically** on page 58
- **View > Assembler > By Address** on page 58

View > Dive

Dives on the selected item. (*Diving* means either displaying the selected information in a window or change the display in this window.) The action that occurs depends upon which item is selected, as follows:

- | | |
|-------------------------|---|
| Stack Trace Pane | Selecting a routine performs a recursive dive operation; that is, it replaces the current contents with the source lines from the selected routine |
| Stack Frame Pane | Opens a Variable Window that contains information about the variable and its contents. |
| Source Pane | <p>If a variable is selected, opens a Variable Window containing information about the variable and its contents. If the line contains a function or subroutine call, TotalView updates the Source Pane so that the line containing the routine is visible.</p> <p>When you dive on a routine, a > indicator appears in the Source Pane's title, indicating that TotalView has performed a nested dive operation. If you again dive, a second > indicator appears. Selecting the < icon to the right of the Source Pane's title <i>unwinds</i> TotalView so that it displays a position it previously displayed.</p> |
| Threads Pane | Selecting an entry performs a dive operation. That is, TotalView replaces the contents of the existing Process Window with information for the selected thread. |

Action Points Pane

Updates the Source Pane so that the line containing the action point is displayed.

In all cases, if the window already exists, TotalView just raises it to the top of the screen.

View > Dive in New Window

The action that occurs depends upon which item is selected, as follows:

Stack Trace Pane Disabled.

Stack Frame Pane

Opens a new Variable Window that contains information about the variable and its contents.

Source Pane Same as **Dive New** if you are diving on a routine name. If you are diving on a variable name, TotalView creates a new Variable Window.

Threads Pane Disabled.

Action Points Pane

Save as **Dive New**.

View > Undive

Pops the Source Pane's *dive stack* so that you return to the place you were previously at in the Source Pane. The dive stack is a history of the source locations you have visited while examining information. This command is analogous to the "Back" button in a browser in that it returns you to a previous position. Each time you "undive", you pop one live off the dive stack. As an alternative, you can select the "<" icon above and to the right of the Source Pane's title.

For additional information, see **View > Redive** on page 54.

View > Redive

View > Redive command:Process window;Process window:View > Redive;Redive command:Process windowPushes the Source Pane's dive stack so that you return to places you "undove" from. The dive stack is a history of the source locations you have visited while examining information. This command is analogous to the "Forward" button in a browser in that it returns you to a position you previously returned from. Each time you "redive", you push one level back onto the dive stack.

As an alternative, you can select the ">" icon above and to the right of the Source Pane's title.

For additional information, see **View > Undive** on page 54.

View > Reset

Resets the source view to the "home" position; that is, it undives the stack and displays the PC.

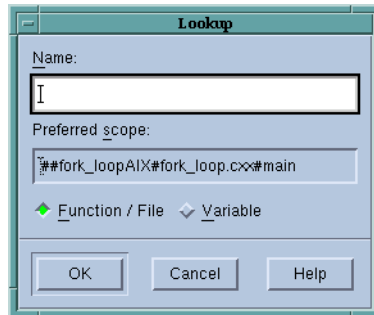
This command lets you undo the effect of command's such as **Edit > Find**, **View > Lookup Variable**, or **View > Lookup Function** or any other command that changes the stack frame. It can also be useful if TotalView does not update the Source Pane.

View > Lookup Function

Use this dialog box to search for a function, file, or variable in your program.

- If you are searching for a function or file, TotalView will display the found information in the Source Pane. If it is not found, TotalView uses a simple spelling correction procedure to search for a function with a similar name.
- If you are searching for a variable, TotalView displays the variable's information in a Variable Window.

Figure 33: View > Lookup
Function Dialog Box



The fields in this dialog box are:

Find	Enter the name of the function or file that TotalView will search for.
Function/File	Searches for a function or file. TotalView assumes that you are typing a function name. If TotalView cannot find the function, it assumes that you are typing a file's name and will search for it. The source for the function is placed into the Source Pane using a dive operation. You can return to the previous contents of the Source Pane by clicking on the < Undive icon in the Source Pane title bar.
Variable	Searches for this variable in your program's symbol table.

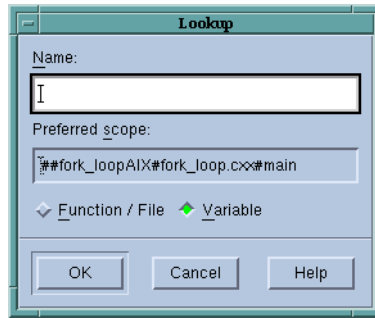
The **Preferred scope** field shows the place from which TotalView begins looking.

View > Lookup Variable

Use this dialog box to search for a variable, function, or file in your program.

- If you are searching for a variable, TotalView displays the variable's information in a Variable Window.
- If you are searching for a function or file, TotalView will display it in the Source Pane. If it is not found, TotalView uses a simple spelling correction procedure to search for a function with a similar name. (See Figure 34.)

Figure 34: View > Lookup Variable Dialog Box



The fields in this dialog box are:

- | | |
|----------------------|---|
| Find | Enter the name of the function or file that TotalView will search for. |
| Function/File | <p>Searches for a function or file. TotalView assumes that you are typing a function name. If TotalView cannot find the function, it assumes that you what you typed was a file's name and will search for it.</p> <p>The source for the function is placed into the Source Pane using a Dive operation. Consequently, you can return to the previous contents of the Source Pane by clicking on the < Undive icon in the Source Pane title bar.</p> |
| Variable | Searches for a local, static, or global variable in your program's symbol table. If you specify a local variable, it must be in the current stack frame. If you specify a pair of addresses instead of a name, TotalView displays the data from the first address to the second (in hex). |

The **Preferred scope** field shows the place from which TotalView begins looking.

If a local and global variable have the same name, TotalView displays the local variable. If TotalView cannot find the local variable, it next checks for a global or static variable.



You cannot tell TotalView which instance of a global or static variable to display. This means that this command cannot locate and display more than one variable with the same name.

If you enter a number, TotalView will display the value at that address as a **<void>**. If you enter two numbers separated by a comma, TotalView displays all locations from the first value to the second as an array of type **<void>**.

After a Variable Window appears, you can edit the type field to show the data in a different way. If you enter an expression, the data type is based on the type of the expression. Similarly, if you enter a cast, the value shown is the result of the cast. Casting is discussed in the *TotalView Users Guide*.

View > Next Process

Replaces the current display with the information for the *next* process.
The *next* process that TotalView displays is the one following this process's entry in the Attached Page of the Root Window.

View > Previous Process

Replaces the current display with the information for the *previous* process.
The *previous* process that TotalView displays is the one preceding this process's entry in the Attached Page of the Root Window.

View > Next Thread

Replaces the current display with the information for the *next* thread.
The *next* thread that TotalView displays is the one following this thread's entry in the Threads Pane.

View > Previous Thread

Replaces the current display with the information for the previous thread.
The *previous* thread that TotalView displays is the one preceding this thread's entry in the Threads Pane.

View > Source As > Source

Tells TotalView that the information displayed within the Source Pane is displayed in the programming language in which it was written.
If this information does not exist, TotalView displays the source as assembler code. If TotalView cannot find the source file, use the **File > Search Path** command to include the directory containing the source file.

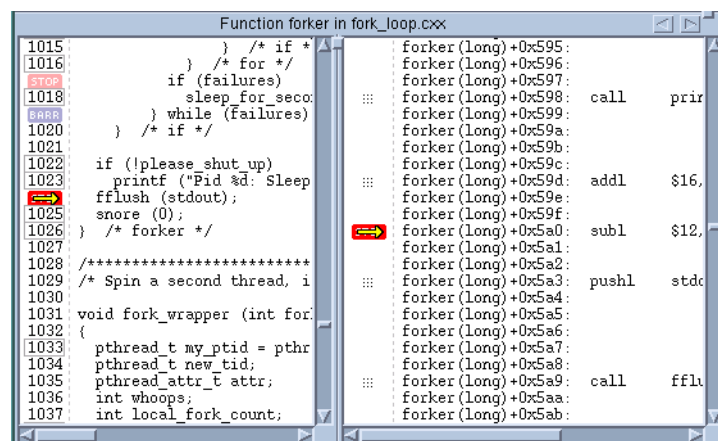
View > Source As > Assembler

Tells TotalView to display the assembler code that the compiler created from your source code.

View > Source As > Both

Tells TotalView to split the Source Pane into two parts and display your source code in the left pane and the assembler code in the right. As you perform actions in one pane, the action is reflected in the other. For example, setting a breakpoint in one pane sets it in both. Similarly, the two move in unison when you step your program.

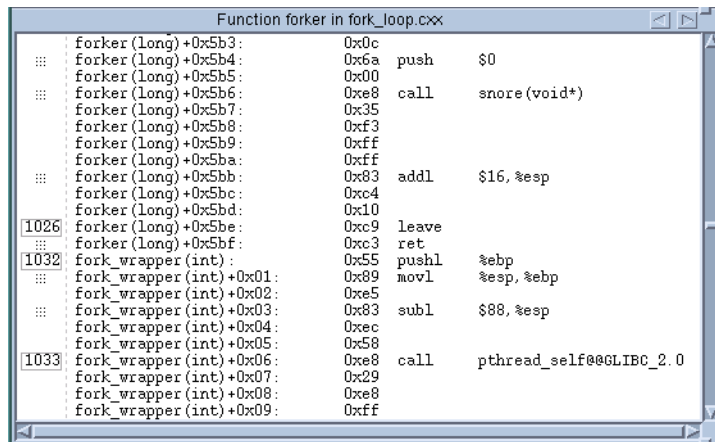
Figure 35: Source Panes: Showing Both



**View > Assembler
> Symbolically**

Tells TotalView that it should display assembler code symbolically. This means that TotalView shows an instruction label and branch target symbolically as a function name plus an offset. (See Figure 36.)

Figure 36: Source Pane:
Showing Assembler Symbolically



```

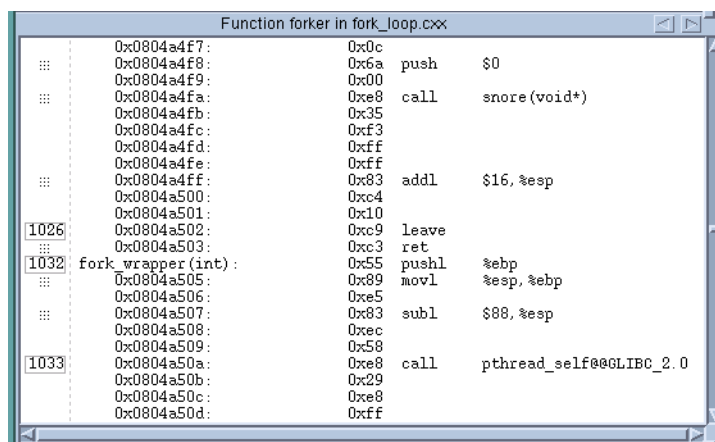
Function forker in fork_loop.cxx
::: forker(long)+0x5b3:      0x0c
::: forker(long)+0x5b4:      0x6a  push    $0
::: forker(long)+0x5b5:      0x00
::: forker(long)+0x5b6:      0xe8  call    snore(void*)
::: forker(long)+0x5b7:      0x35
::: forker(long)+0x5b8:      0xf3
::: forker(long)+0x5b9:      0xff
::: forker(long)+0x5ba:      0xff
::: forker(long)+0x5bb:      0x83  addl    $16, %esp
::: forker(long)+0x5bc:      0xc4
::: forker(long)+0x5bd:      0x10
1026 forker(long)+0x5be:      0xc9  leave
::: forker(long)+0x5bf:      0xc3  ret
1032 fork_wrapper(int):      0x55  pushl   %ebp
::: fork_wrapper(int)+0x01:    0x89  movl    %esp, %ebp
::: fork_wrapper(int)+0x02:    0xe5
::: fork_wrapper(int)+0x03:    0x83  subl    $88, %esp
::: fork_wrapper(int)+0x04:    0xec
::: fork_wrapper(int)+0x05:    0x58
1033 fork_wrapper(int)+0x06:    0xe8  call    pthread_self@@GLIBC_2.0
::: fork_wrapper(int)+0x07:    0x29
::: fork_wrapper(int)+0x08:    0xe8
::: fork_wrapper(int)+0x09:    0xff

```

**View > Assembler
> By Address**

Tells TotalView that it should display assembler code by address. This means that TotalView shows the instruction labels and branch targets as hexadecimal addresses. TotalView will, however, always show the target address of branch-to-subroutine instructions symbolically. (See Figure 37.)

Figure 37: Source Pane:
Showing Assembler By Address



```

Function forker in fork_loop.cxx
::: 0x0804a4f7:      0x0c
::: 0x0804a4f8:      0x6a  push    $0
::: 0x0804a4f9:      0x00
::: 0x0804a4fa:      0xe8  call    snore(void*)
::: 0x0804a4fb:      0x35
::: 0x0804a4fc:      0xf3
::: 0x0804a4fd:      0xff
::: 0x0804a4fe:      0xff
::: 0x0804a4ff:      0x83  addl    $16, %esp
::: 0x0804a500:      0xc4
::: 0x0804a501:      0x10
1026 0x0804a502:      0xc9  leave
::: 0x0804a503:      0xc3  ret
1032 fork_wrapper(int):      0x55  pushl   %ebp
::: 0x0804a505:      0x89  movl    %esp, %ebp
::: 0x0804a506:      0xe5
::: 0x0804a507:      0x83  subl    $88, %esp
::: 0x0804a508:      0xec
::: 0x0804a509:      0x58
1033 0x0804a50a:      0xe8  call    pthread_self@@GLIBC_2.0
::: 0x0804a50b:      0x29
::: 0x0804a50c:      0xe8
::: 0x0804a50d:      0xff

```


Group Menu Commands

The following commands are on the Group pulldown:

- **Group > Go** on page 59
- **Group > Halt** on page 60
- **Group > Next** on page 60
- **Group > Step** on page 60
- **Group > Out** on page 61
- **Group > Run To** on page 61
- **Group > Next Instruction** on page 62
- **Group > Step Instruction** on page 62
- **Group > Share Submenu** on page 63
- **Group > Workers Submenu** on page 66
- **Group > Lockstep Submenu** on page 69
- **Group > Hold** on page 72
- **Group > Release** on page 72
- **File > Rescan Libraries** on page 50
- **Group > Attach Subset** on page 72
- **Group > Edit Group** on page 74
- **Group > Restart** on page 76
- **Group > Delete** on page 76

All commands in this group operate at group width. If the command causes stepping to occur, the stepping is focused on the control group of the thread of interest.

Within the CLI, this is equivalent to executing a command having a focus of **gC**.

Group > Go

Starts or continues all processes in all control groups of the thread of interest.

Contrast the action performed by this command with the process and thread versions:

- The process version starts or continues all threads in the process of interest.
- The thread version continues only the current thread; the other threads in the process remain in their current state.

Typing **Group > Go** or **Process > Go** creates the process if you have not yet started executing your program.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Halt

Halts all processes in the control group of the thread of interest. After this occurs, TotalView updates all windows associated with threads in this group.

Note the difference between this command and Process > Halt. In a multi-process program, this command stops all the processes in the current control group. **Process > Halt** stops all of the threads in the process of interest.

Group > Next

"Next steps" all processes in the control group that are in the same lockstep group as the thread of interest over a source line. That is, TotalView lets these lockstep threads execute one source line. If the line contains function calls, however, TotalView executes the entire function as if it were a single statement.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread arrive at the next source statement or some thread hits a breakpoint or encounter an error.



If more than one statement exists on the line, all are executed.

Contrast this command with the process and thread versions:

- The process version steps all threads in the process of interest.
- The threads version only steps the current thread; the other threads in the process remain stopped during the step.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Step

Steps all processes in the control group that are in the same lockstep group as the thread of interest. That is, TotalView lets these lockstep threads execute one source line. If the current line contains a function call, TotalView steps into this function.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread arrive at the next source statement.



If more than one statement exists on the line, all are executed.

Contrast this command with the process and thread versions:

- The process version steps all threads in the process of interest.
- The threads version only steps the current thread; the other threads in the process remain stopped during the step.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Out

Continues all processes in the control group that are in the same lockstep group as the thread of interest until execution returns from the current function.

You can tell TotalView to return out of more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine A in the Stack Trace Pane. In this case, the selected routine name is the routine you want to return to.



Be careful to distinguish between this command and "Run To". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread return from the function or some thread hits a breakpoint.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process.
- The thread version runs only the current thread; the other threads in the process remain stopped.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Run To

Continues all processes in the thread of interest's control group until one thread in each process in the thread of interest's lockstep group reaches the selected source line or instruction. While these threads are executing, other threads in all other control groups run freely.



*If some process never reaches the selection, TotalView keeps waiting; if this happens, select **Group > Halt** to interrupt the operation.*

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in a another stack frame, in which case the

operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process.
- The thread version runs only the current thread; the other threads in the process remain stopped.



Be careful to distinguish between this command and "Out". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Next Instruction

"Next steps" all processes in the control group that are in the same lockstep group as the thread of interest over an assembler instruction. That is, TotalView lets these lockstep threads execute one source line. If the line contains a function call, TotalView executes the entire function as if it were a single statement.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to run freely. It then waits until the thread of interest and each matching thread reach this instruction.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process of interest.
- The thread version runs only the current thread; the other threads in the process remain stopped.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the TotalView Users Guide.

Group > Step Instruction

Steps all processes in the control group that are in the same lockstep group as the thread of interest over one assembler instruction. That is, TotalView lets these lockstep threads execute one source line. If the line contains a function call, TotalView steps into this function.

TotalView examines the control group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes in all control groups to

run freely. It then waits until the thread of interest and each matching thread reach this instruction.

Contrast the action performed by this command with the thread and group version:

- The process version runs all threads in the process of interest.
- The thread version runs only the current thread; the other threads in the process remain stopped.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share Submenu

The commands on this submenu are:

- Group > Share > Go on page 63
- Group > Share > Halt on page 63
- Group > Share > Next on page 64
- Group > Share > Step on page 64
- Group > Share > Out on page 64
- Group > Share > Run To on page 65
- Group > Share > Next Instruction on page 65
- Group > Share > Step Instruction on page 66

The commands on this command apply to the share group associated with the current group. This thread is call the thread of interest. The share group contains the related processes that share the same source code. A share group contains all members of a control group that share the same executable image. (Note, however, that dynamically loaded libraries may vary between share groups member.)

TotalView automatically places processes in share groups based on their program group and their executable image. You can't change a share group's members.

This is equivalent to executing a command within the CLI having a focus of gS.

Group > Share > Go

Continues all processes in the share group associated with the thread of interest. These processes are within the thread of interest's control group.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share > Halt

Halts all processes in the share group associated with the thread of interest. After this occurs, TotalView updates all windows associated with the processes it just stopped.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share > Next

"Next steps" all processes in the share group that are in the same lockstep group as the thread of interest over a source line. That is, TotalView lets these lockstep threads execute one source line. If the line contains function calls, TotalView executes the entire function as if it were a single statement.

TotalView examines the share group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this statement or some thread hits a breakpoint.

TotalView allows all threads in your share group to run freely until the thread of interest and the other threads in the same share group reach the next source line.



If more than one statement exists on the line, all are executed.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share > Step

Steps all processes in the share group that are in the same lockstep group as the thread of interest over a source line. That is, TotalView lets these lockstep threads execute one source line. If the line contains function calls, TotalView steps into the function.

TotalView examines the share group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all other processes to run freely. It then waits until the thread of interest and each matching thread reach this statement or some thread hits a breakpoint.



If more than one statement exists on the line, all are executed.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share > Out

Continues all processes in the share group that are in the same lockstep group as the thread of interest until execution returns from the current location in the function in which it is executing.

You can tell TotalView to return out of more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine A in the Stack Trace Pane. In this case, the selected routine name is the routine you want to return *to*.



Be careful to distinguish between this command and "Run To". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

TotalView examines the share group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each process in the share group, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread returns from the current function or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share > Run To

Continues all processes in the current share group until one thread in each process in the share group of the thread of interest reaches the selected source line or instruction.



*If some process never reaches the selection, TotalView keeps waiting. TotalView should popup a small dialog box with a cancel button that allows you to stop processing. If this window does not appear, select the **Group > Share > Halt** command to interrupt the operation.*

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in a another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.



Be careful to distinguish between this command and "Out". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share > Next Instruction

"Next steps" all processes in the share group that are in the same lockstep group as the thread of interest over an assembler instruction. That is, TotalView lets these lockstep threads execute one instruction line. If the line contains a function call, TotalView executes the entire function as if it were a single instruction.

TotalView examines the share group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this instruction or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Share > Step Instruction

Steps all processes in the share group that are in the same lockstep group as the thread of interest by one assembler instruction. That is, TotalView lets these lockstep threads execute one source line. If the current instruction is a function call, TotalView steps into that function. Other threads in the share group run freely.

TotalView examines the share group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this instruction or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers Submenu

The commands on this submenu are:

- Group > Workers > Go on page 66
- Group > Workers > Halt on page 66
- Group > Workers > Next on page 67
- Group > Workers > Step on page 67
- Group > Workers > Out on page 67
- Group > Workers > Run To on page 68
- Group > Workers > Next Instruction on page 69
- Group > Workers > Step Instruction on page 69

The command on this menu affect the workers group. Contains all worker threads from all processes in the control group. By default, it contains all threads except the kernel-level manager threads that can be identified. You can use all group manipulation commands on workers group. However, you cannot delete them.

This is equivalent to executing a command within the CLI having a focus of gW.

Group > Workers > Go

Continues all threads in the workers group associated with the thread of interest.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers > Halt

Halts all threads in the workers group associated with the thread of interest. After this occurs, TotalView updates Processes Windows associated with these threads.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers > Next

"Next steps" all threads in the lockstep group associated with the thread of interest that are within the thread of interest's workers group. That is, TotalView lets these lockstep threads execute one source line. If the line contains function calls, TotalView executes the entire function as if it were a single statement.

TotalView examines the workers group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this source line or some thread hits a breakpoint.



If more than one statement exists on the line, all are executed.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers > Step

Steps all threads in the lockstep group associated with the thread of interest that are within the thread of interest's workers group. That is, TotalView lets these lockstep threads execute one source line. If the current line is a function call, TotalView steps into this function.

TotalView examines the workers group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this source line or some thread hits a breakpoint.



If more than one statement exists on the line, all are executed.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers > Out

Continues all threads in the lockstep group associated with the thread of interest that are within the thread of interest's workers group until the thread of interest and all other threads in its lockstep group return from the function in which it is executing.

You can tell TotalView to return out of more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine A in the Stack Trace

Pane. In this case, the selected routine name is the routine you want to return to.



Be careful to distinguish between this command and "Run To". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

TotalView examines the workers group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this instruction or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers > Run To

Continues all threads in the lockstep group associated with the thread of interest that are within the thread of interest's workers group until the thread of interest reaches the selected source line or instruction.



*If some process never reaches the selection, TotalView keeps waiting. TotalView should popup a small dialog box with a cancel button that allows you to stop processing. If this window does not appear, select the **Group > Workers > Halt** command to interrupt the operation.*

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in a another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.



Be careful to distinguish between this command and "Out". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

TotalView examines the workers group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach the selected line or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers > Next Instruction

"Next steps" all threads in the lockstep group associated with the thread of interest that are contained within the thread of interest's workers group. That is, TotalView lets these lockstep threads execute one source line. If the line contains function calls, TotalView executes the entire function as if it were a single statement.

TotalView examines the workers group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this instruction or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Workers > Step Instruction

Steps all threads in the lockstep group associated with the thread of interest that are contained within the thread of interest's workers group. That is, TotalView lets these lockstep threads execute one source line. If the line contains function calls, TotalView steps into that function.

TotalView examines the workers group to identify which processes have a thread stopped at the same location as the thread of interest. This thread is called the *matching* thread. After selecting a matching thread from each matching process, TotalView allows all processes to run freely. It then waits until the thread of interest and each matching thread reach this instruction or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep Submenu

The commands on this submenu are:

- Group > Lockstep > Go on page 70
- Group > Lockstep > Halt on page 70
- Group > Lockstep > Next on page 70
- Group > Lockstep > Step on page 70
- Group > Lockstep > Out on page 70
- Group > Lockstep > Run To on page 71
- Group > Lockstep > Next Instruction on page 71
- Group > Lockstep > Step Instruction on page 71

The commands on this submenu are group level lockstep command. The lockstep group contains every stopped thread in a share group that have the same PC. There is one lockstep group for every thread.

This is equivalent to executing a CLI command with a focus of **gL**.

Group > Lockstep > Go

Continues all threads in the lockstep group associated with the thread of interest.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep > Halt

Halts all processes in the control group of the thread of interest. This function exists solely so that you do not have to change contexts to halt running processes. After these processes are halted, TotalView updates all process windows associated with these threads.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep > Next

"Next steps" all processes in the lockstep group associated with the current thread over a source line. That is, this command tells TotalView to step over any function calls to the next source line in the current function.

TotalView allows all processes to run freely until the lockstep group reaches this source line or some thread hits a breakpoint.



If more than one statement exists on the line, all are executed.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep > Step

Steps all processes in the lockstep group associated with the current thread to the next source line statement. If the current line is a function call, TotalView steps into this function.

TotalView allows all processes to run freely until the lockstep group reaches this source line or some thread hits a breakpoint.



If more than one statement exists on the line, all are executed.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep > Out

Continues the lockstep group associated with the thread of interest until this thread and all other threads in its lockstep group return from the function in which it is executing.

TotalView allows all processes to run freely until the lockstep group returns from this routine or some thread hits a breakpoint.

You can tell TotalView to return out of more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can return to routine A by selecting routine A in the Stack Trace

Pane. In this case, the selected routine name is the routine you want to return to.



Be careful to distinguish between this command and "Run To". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep > Run To

Tells TotalView that it should run threads in the lockstep group until these threads reach a target line or instruction.

TotalView allows all processes to run freely until the lockstep group returns from this routine or some thread hits a breakpoint.



*If some process never reaches the selection, TotalView keeps waiting. TotalView should popup a small dialog box with a cancel button that allows you to stop processing. If this window does not appear, select the **Group > Lockstep > Halt** command to interrupt the operation.*

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in a another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.



Be careful to distinguish between this command and "Out". The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep > Next Instruction

"Next steps" all processes in the lockstep group over an assembler instruction. That is, this command tells TotalView to step over a function calls to the next instruction in the current routine.

TotalView allows all processes to run freely until the lockstep group returns from this routine or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Group > Lockstep > Step Instruction

Steps all processes in the lockstep group by one assembler instruction. That is, this command causes one assembler instruction to be executed for all threads in the thread of interest's lockstep group. If the current instruction is a function call, TotalView steps into that function.

TotalView allows all processes to run freely until the lockstep group returns from this routine or some thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see *"Using Groups, Processes, and Threads"* in the *"TotalView Users Guide"*.

Group > Hold

Holds all processes in the control group of the thread of interest. After using this command, you will need to explicitly release the processes before they can again execute.

If all of the processes in the control group are not currently held, this command stops and holds them.

For more information, see the **"Group > Release"** on page 72.

For more information on processes and threads and their behavior while being stepped, see *"Using Groups, Processes, and Threads"* in the *"TotalView Users Guide"*.

Group > Release

Releases all processes. "Releasing" means that TotalView will allow the process to execute if a command tell it to. That is, this command does not continue the group—you must use a separate "Go" command for that.

For more information on processes and threads and their behavior while being stepped, see *"Using Groups, Processes, and Threads"* in the *"TotalView Users Guide"*.

Group > Attach Subset

Lets you indicate which processes TotalView should attach to when these processes begin executing. Limiting the processes to which TotalView attaches is beneficial as TotalView does not have to be concerned with unattached processes. That is, because you know that you will not be interested in a what goes on in within a process, you can cut down on the time that TotalView uses to attach to all or most of your processes.

Processes to Attach To

Use the controls in this area to specify the processes to which TotalView should attach when they are created. You have three choices:

Selection Area

Individually select or deselect processes

All Attach to all of the listed processes.

None Do not attach to any of these processes.

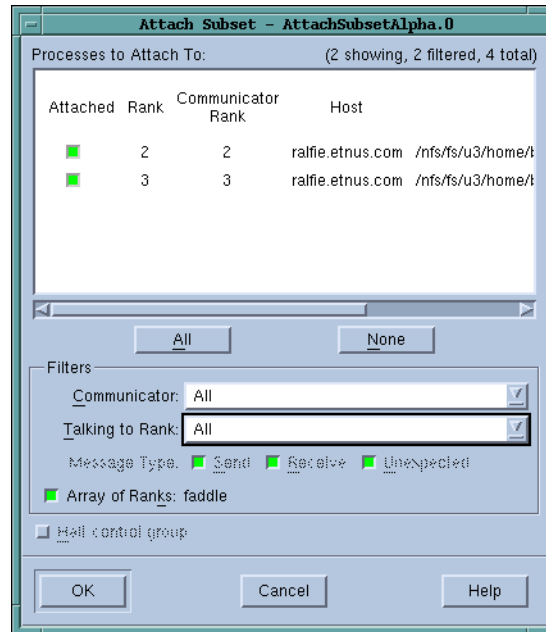
After selecting **All** or **None**, you can individually select or unselect processes. That is, if you only want to select a couple of processes, begin by clicking **None**, then select the few to which TotalView should attach.

Filters You can restrict the list by selecting the controls in this area.

Communicator

The communicators within this list tell TotalView which processes it should display. Selecting one of

Figure 38: Attach Subset Dialog Box



the communicators contained within this list tells TotalView that it should only display processes using this communicator. You can then select or clear these values in one of the three ways just discussed.

Talking to Rank

TotalView will limit the graph to communicators that receive messages from the indicated ranks. In addition to your rank numbers, TotalView includes two special variables: **All** and **MPI_ANY_SOURCE**.

Message Type

TotalView will only show **Send**, **Receive**, or **Unrestricted** messages.

Array of Ranks

This checkbox is automatically selected by TotalView if you have invoked this command from the **Tools > Attach Subset (Array of Ranks)** command. If the Variable Window is displaying an array, invoking this command tells TotalView that the array's elements indicate ranks.

Halt control group

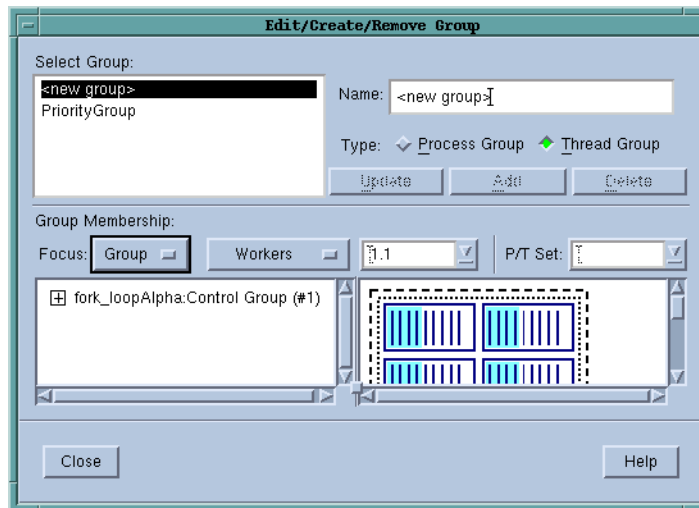
Selecting this button tells to stop all of the processes in the current process's control group after it attaches to a process. If it isn't selected, TotalView will immediately execute the control group after it attaches to them.

Group > Edit Group

Allows you to create, delete, and edit P/T set groups. The control is divided into two parts. The top part is where you manage the group's identify. The bottom part allows you to specify which processes and threads will become part of the group.

The controls in the top part of the dialog box let you update, add, and delete groups. If you are updating or creating a group, the **Type** radio buttons indicate if this is a processor thread group. (See Figure 39.)

Figure 39: Group > Edit Group Dialog Box



Update Here's how to update a group:

- 1 After selecting a group in the left hand list, make your changes. You'll be able to select process and threads in the P/T selector (the third pulldown) and edit the existing set by entering an expression in the Expression area (the right text control).
- 2 Click the **Update** button.

Add Here's how to add a group:

- 1 Begin by selecting **<new_group>** in the left-hand list if it isn't already displayed in the **Name** field and enter the group's name.
- 2 Selecting either the **Process Group** or **Thread Group** radio button.
- 3 Use the controls in the bottom part of the window to select the processes and threads that will be part of the group.
- 4 Click the **Add** button.

Delete Here's how to delete a group:

- 1 Select the group being deleted in the left-hand list.
- 2 Click the **Delete** button.

Defining Groups

You will need to have a background in managing asynchronous threads. This information is discussed in Chapters 2 and 11 of the TotalView Users Guide.

Group Membership areas

- The first pulldown, which is called the *Width Pulldown*, has four elements on it: **All**, **Group**, **Process**, and **Thread**. Your choices here indicate the width of the command. The *Width Pulldown* tells TotalView where it should look when it tries to determine what it will be manipulating. For example, if **Group** is selected, a **Go** command continues the group. Which group TotalView will continue is set by the choices on the second pulldown.
- The second pulldown, which is called the *Group Pulldown*, tells TotalView which processes and threads within the scope defined by the *Width Pulldown* it should manipulate. For example, you could tell TotalView to step the threads defined within the current workers group that are contained within the current process. The elements on this pulldown are **Control**, **Share**, **Workers**, and **Lockstep**.
If you have created your own groups, their names will appear on this pulldown.
- The *P/T Selector* (the pulldown box in the middle) lets you change the focus of the action from the currently defined process and thread to any other process and thread that TotalView controls.

What is selected when you set these controls from within the GUI or specify a focus using the CLI can get quite complicated.

Expression Area

The expression area lets you create more complicated sets and to edit an existing group. While you can specify threads using and processes in this area, you can also use the following operators to aggregate collections of threads:

breakpoint()	Returns a list of all threads that are stopped at a breakpoint.
error()	Returns a list of all threads stopped due to an error.
existent()	Returns a list of all threads.
held()	Returns a list of all threads that are held.
nonexistent()	Returns a list of all processes that have exited or which, while loaded, have not yet been created.
running()	Returns a list of all running threads.
stopped()	Returns a list of all stopped threads.
unheld()	Returns a list of all threads that are not held.
watchpoint()	Returns a list of all threads that are stopped at a watchpoint.

The argument that all of these operators use is a P/T set. You specify this set in the same way that a P/T set is specified for the **dfocus** command. For example, **watchpoint(L)** returns all threads in the current lockstep group.

You can treat the lists returned by these operators as list using the following three operators:

	Creates a union; that is, all members of the sets.
-	Creates a difference; that is, all members of the first set that are not also members of a second set.
&	Creates an intersection; that is, all members of the first set that are also members of the second set.

The dot (.) operator, which indicates the current set, can be helpful when you are editing an existing set.

For examples of using these operators, see Chapter 11 of the TotalView Users Guide.

Group > Restart

Deletes (kills) all processes in the current control group, refocuses the Process Window on the master process, and then creates and starts this process. This command is equivalent to **Group > Delete** followed by a **Process > Go**.

Group > Delete

Deletes (kills) all processes in the current control group. TotalView focuses the Process Window on the master process.

The next time you start the program with the **Process > Go** command, TotalView creates and starts a new master process.

Process Menu Commands

The following commands appear on the **Process** pulldown:

- **Process > Go** on page 77
- **Process > Halt** on page 77
- **Process > Next** on page 77
- **Process > Step** on page 77
- **Process > Out** on page 78
- **Process > Run To** on page 78
- **Process > Next Instruction** on page 79
- **Process > Step Instruction** on page 79
- **Process > Workers Submenu** on page 80
- **Process > Lockstep Submenu** on page 83
- **Process > Hold** on page 86
- **Process > Hold Threads** on page 86
- **Process > Release Threads** on page 86
- **Process > Create** on page 87
- **Process > Detach** on page 87
- **Process > Startup Parameters** on page 87

All commands in this group operate on the current process. If the command causes stepping to occur, the stepping is focused on the control group of the thread of interest. A control group includes children that were forked (processes that share the same source code as the parent) and children that were forked but which subsequently called **execve()**. That is, a control group includes the children of the created processes that do not share the same source code as the parent.

Within the CLI, this is equivalent to executing a command having a focus of **pC**.

Process > Go

Starts or continues the current process. This command starts all threads in the process, not just the current thread.

Contrast the action performed by this command with the thread and group version:

- The group version runs all processes in the control group containing the thread of interest.
- The thread version runs only the thread of interest; the other threads in the process remain stopped during the step.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Halt

Halts the process. This command stops all executing threads in the process of interest. After they are stopped, TotalView updates the windows in which information about the process or its threads appears.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Next

"Next-steps" all threads in the lockstep group associated with the thread of interest that are within the current process over a source line. That is, TotalView lets these threads execute one source line. If the line contains function calls, totalView executes the entire function as if it were a single statement.



If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has source line information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has source line information.

While the lockstep group is being run to the next source line, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping, or another thread hits a breakpoint.

For related information, see "Process > Go" on page 77.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Step

Steps all threads in the lockstep group associated with the thread of interest that are within the process. That is, TotalView lets these lockstep threads execute one source line. If the current line contains a function call, TotalView steps into it.



If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has source line information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has source line information.

While the lockstep group is being run to the next source line, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping, or another thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Out

Continues all threads in the lockstep group associated with the thread of interest that are within the process until the current thread returns from the function in which it is executing.

You can tell TotalView to return from more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine B in the Stack Trace Pane. (The selected routine name is the routine you want to return *from*.)

While the lockstep group is being run to the next source line, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group return from the routine or another thread hits a breakpoint.

For related information, see "Process > Go" on page 77.



Be careful to distinguish between this command and Run To. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Run To

Continues all threads in the lockstep group associated with the thread of interest that are within the process. The processes continue running until threads in the lockstep group reach the selected source line or instruction in the Source Pane.

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in a another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.

While the lockstep group is being run to the next source line or instruction, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping or another thread hits a breakpoint.



Be careful to distinguish between this command and Out. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For related information, see "**Process > Go**" on page 77.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Next Instruction

"Next-steps" all threads in the lockstep group associated with the thread of interest that are within the current process over an assembler instruction. That is, this command tells TotalView to allow execution of the next instruction. If the current instruction is a function call, the call is executed as if it were a single instruction.



If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has information.

While the lockstep group is being run to the next instruction, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping or another thread hits a breakpoint.

For related information, see "**Process > Go**" on page 77.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Step Instruction

Steps all threads in the lockstep group associated with the thread of interest that are within the thread of interest's process over an assembler instruction. That is, TotalView lets one assembler instruction be executed. If the current instruction is a function call, TotalView steps into that function.

While the lockstep group is being run to the next instruction, the entire process is also allowed to run. The operation does not complete until all the threads in the lockstep group thread finish stepping or another thread hits a breakpoint.

Contrast this command with the thread and group equivalents:

- The thread version only steps the current thread.
- The group version steps all matching processes in the group.



If more than one statement exists on the line, all are executed.

If the process exists, TotalView runs the selected thread until the PC reaches the next address for which it has information. If the process does not yet exist, this command launches the process and runs the first thread until the PC reaches the first address for which TotalView has information.

For related information, see “**Process > Go**” on page 77.

For more information on processes and threads and their behavior while being stepped, see “Using Groups, Processes, and Threads” in the “TotalView Users Guide”.

Process > Workers Submenu

The commands on this submenu are:

- **Process > Workers > Go** on page 80
- **Process > Workers > Halt** on page 80
- **Process > Workers > Next** on page 80
- **Process > Workers > Step** on page 81
- **Process > Workers > Out** on page 81
- **Process > Workers > Run To** on page 82
- **Process > Workers > Next Instruction** on page 82
- **Process > Workers > Step Instruction** on page 82

Process > Workers > Go

Continues all of the worker threads within a process.

For more information on processes and threads and their behavior while being stepped, see “Using Groups, Processes, and Threads” in the “TotalView Users Guide”.

Process > Workers > Halt

Halts all of the process’s worker threads. After they are stopped, TotalView updates the windows associated with these threads.

For more information on processes and threads and their behavior while being stepped, see “Using Groups, Processes, and Threads” in the “TotalView Users Guide”.

Process > Workers > Next

“Next-steps” all worker threads within a process that are in the thread of interest’s lockstep group. That is, TotalView executes one line in each of these threads. If the line contains a function call, TotalView executes the call as if it were a single statement.

It runs the process until the current thread’s PC reaches the next executable source line.



If more than one statement exists on the line, all are executed.

While the workers group's threads are being run to the next source line, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.

For related information, see "**Process > Go**" on page 77.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Workers > Step

Steps all worker threads within a process that are in the thread of interest's lockstep group. That is, TotalView executes one line in each of these threads. If this line contains a function call, TotalView steps into it.



If more than one statement exists on the line, all are executed.

While the workers group's threads are being run to the next source line, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Workers > Out

Continues all worker threads within a process that are in the thread of interest's lockstep group until execution control returns from the function in which execution had stopped.

You can tell TotalView to return from more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine A in the Stack Trace Pane. In this case, the selected routine name is the routine you want to return to.



Be careful to distinguish between this command and Run To. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

While the workers group's threads are being run out of the function, the entire process is also allowed to run. The operation completes when the threads being stepped reach their target or another thread hits a breakpoint.

For more information

, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Workers > Run To

Continues all worker threads within a process that are in the thread of interest's lockstep group until execution control reaches the selected source line or instruction.

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.



Be careful to distinguish between this command and Out. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

While the workers group's threads are being run to the selected source line or instruction, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.

For related information, see "**Process > Go**" on page 77.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Workers > Next Instruction

"Next-steps" all worker threads within a process that are in the thread of interest's lockstep group. That is, TotalView executes one instruction in each of these threads. If the line contains a function call, TotalView executes the call as if it were a single statement.

While the workers group's threads are being run to the next instruction, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.



If more than one statement exists on the line, all are executed.

For related information, see "**Process > Go**" on page 77.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Workers > Step Instruction

Steps all worker threads within a process that are in the thread of interest's lockstep group. That is, TotalView executes one instruction in each of these threads. If the line contains a function call, TotalView steps into the function.

While the workers group's threads are being run to the next instruction, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.



If more than one statement exists on the line, all are executed.

For related information, see “**Process > Go**” on page 77.

For more information on processes and threads and their behavior while being stepped, see “*Using Groups, Processes, and Threads*” in the “*TotalView Users Guide*”.

Process > Lockstep Submenu

The commands on this submenu are:

- **Process > Lockstep > Go** on page 83
- **Process > Lockstep > Halt** on page 83
- **Process > Lockstep > Next** on page 83
- **Process > Lockstep > Step** on page 84
- **Process > Lockstep > Out** on page 84
- **Process > Lockstep > Run To** on page 84
- **Process > Lockstep > Next Instruction** on page 85
- **Process > Lockstep > Step Instruction** on page 85

Process > Lockstep > Go

Continues all of the threads within the process that are in the thread of interest’s lockstep group.



This command starts all threads in the process, not just the current thread.

For more information on processes and threads and their behavior while being stepped, see “*Using Groups, Processes, and Threads*” in the “*TotalView Users Guide*”.

Process > Lockstep > Halt

Halts the control group of the thread of interest. This function exists solely so that you do not have to change contexts to halt running processes. After these processes are halted, TotalView updates all process windows associated with these threads.

For more information on processes and threads and their behavior while being stepped, see “*Using Groups, Processes, and Threads*” in the “*TotalView Users Guide*”.

Process > Lockstep > Next

“Next-steps” all lockstep threads within a process that are in the thread of interest’s lockstep group. That is, TotalView executes one line in each of these threads. If the line contains a function call, TotalView executes the call as if it were a single statement.



If more than one statement exists on the line, all are executed.

While the lockstep group’s threads are being run to the next source line, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a break-point.

For related information, see “**Process > Go**” on page 77.

For more information on processes and threads and their behavior while being stepped, see “*Using Groups, Processes, and Threads*” in the “*TotalView Users Guide*”.

Process > Lockstep > Step

Steps all lockstep threads within a process that are in the thread of interest's lockstep group. That is, TotalView executes one line in each of these threads. If the line contains a function call, TotalView executes the call as if it were a single statement.



If more than one statement exists on the line, all are executed.

While the lockstep group's threads are being run to the next source line, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Lockstep > Out

Continues all lockstep threads within a process that are in the thread of interest's lockstep group until execution control returns from the function in which execution had stopped.

You can tell TotalView to return from more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can return to routine A by selecting routine A in the Stack Trace Pane. In this case, the selected routine name is the routine you want to return to.



Be careful to distinguish between this command and Run To. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

While the lockstep group's threads are being run out of the function, the entire process is also allowed to run. The operation completes when the threads being stepped reaches this goal or another thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Process > Lockstep > Run To

Continues all lockstep threads within a process that are in the thread of interest's lockstep group until execution control reaches the selected line or instruction.

This command behaves like a temporary breakpoint. It differs in that it is bound to a stack frame.

While the lockstep group's threads are being run to the next source line or instruction, the entire process is also allowed to run. The operation completes when the threads being stepped reach the selected line or instruction or another thread hits a breakpoint.

Before using this command, you must select a line in the Source Pane. You can also select and run to a line in another stack frame, in which case the operation does not complete until the thread reaches the selected line. If you are running to a line in another stack frame, TotalView will change the stack display because execution has moved to a different.



Be careful to distinguish between this command and Out. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For related information, see "**Process > Go**" on page 77.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Lockstep > Next Instruction

"Next-steps" all lockstep threads within a process that are in the thread of interest's lockstep group. That is, TotalView executes one instruction in each of these threads. If the instruction contains a function call, TotalView executes the call as if it were a single statement.

While the lockstep group's threads are being run to the next instruction, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.

For related information, see "**Process > Go**" on page 77.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Lockstep > Step Instruction

Steps all lockstep threads within a process that are in the thread of interest's lockstep group. That is, TotalView executes one instruction in each of these threads. If the instruction contains a function call, TotalView executes the call as if it were a single statement.

TotalView waits while your program executes the instruction; nothing else can be done until the step completes. As with the source-level step command, this command runs the entire process while the current thread is being stepped.

While the lockstep group's threads are being run to the next source line or instruction, the entire process is also allowed to run. The operation completes when the threads being stepped finish stepping or another thread hits a breakpoint.

For more information on processes and threads and their behavior while being stepped, see "*Using Groups, Processes, and Threads*" in the "*TotalView Users Guide*".

Process > Hold

Tells to TotalView to hold or release the current process.

- *Hold* means that you are telling TotalView that when you use a command that would run the threads in this process (such as a **Group > Go**), TotalView should ignore the command.
- *Release* means that you are telling TotalView that the thread can execute when you use a run command such as **Group > Go**.

The **Process > Hold Threads** on page 86 command can also hold. The description for that command describes the differences.

If the process is not currently held, this command stops and holds it. If it is currently held, this command releases it (but it does not continue the process—you must use a separate **Go** command for that).

For more information on processes and threads and their behavior while being stepped, see “Using Groups, Processes, and Threads” in the “TotalView Users Guide”.

Process > Hold Threads

Tells to TotalView to hold all threads in the current process. *Hold* means that you are telling TotalView that when you use a command that would run the threads in this process (such as a **Group > Go** command), it is ignored. Note that TotalView does not allow you to hold manager threads.

If a thread is already held when you enter this command, this state will be remembered when you enter a **Process > Release Threads** on page 86 command.

Compare this command with the **Process > Hold** on page 86 command. If you hold a process, none of the threads will ever run; that is, a *go* has no effect. When the process is held, the hold state of any of the process’s threads is ignored.

If you release the process and hold all of the threads, the effect is essentially the same: none of the non-manager threads will run; and a *go* command will only run manger threads.

Here’s one example illustrating why you need both commands. Suppose a thread barrier is not yet satisfied: some threads are held at the barrier and some are expected to get there later. Since a thread barrier can span processes, you can have multiple processes with multiple threads trying to get to the barrier. You may want to hold one or more processes while allowing threads in other processes to get to the barrier. In this case, holding at the process level means that you are not destroying the hold state of the individual threads. At a later time, you can release the processes to allow the remaining threads to get to the barrier.

For more information on processes and threads and their behavior while being stepped, see “Using Groups, Processes, and Threads” in the “TotalView Users Guide”.

Process > Release Threads

Releases all threads that were held in the process. See **Process > Hold Threads** on page 86 for more information.

Process > Create

Create a process without starting it. If a program is linked with shared libraries, TotalView allows the dynamic loader to map into these libraries. Creating a process without starting it is useful if you need to:

- Create action points, watchpoints, or change global variables after a process is created, but before it runs.
- Debug C++ static constructor code.

Process > Detach

Detaches from the current process. Once detached, the process is no longer under TotalView control.

If you want to stop the process's execution, you should also use a **Thread > Continuation Signal** command to send a signal to the process. If you set the continuation signal to **SIGSTOP**, the process is stopped after you detach from it. Note that you can only send a signal to a thread; you cannot detach from an individual thread.

Use **Group > Attach Subset** to detach from one or more processes in a parallel program.

Process > Startup Parameters

Use this dialog box to set:

- Command arguments
- Environment variables
- Input and Output file names

In all cases, the information you enter or change in this dialog box is not used until TotalView starts a process.

Arguments Page

Contains the arguments that TotalView passes to a process when it is next launched. These arguments are usually entered using TotalView's **-a** option. If you enter them this way, using this page allows you to alter them. If you do not use the **-a** option, this page allows you to name the arguments that TotalView passes to the process the next time it is started. (See Figure 40.)

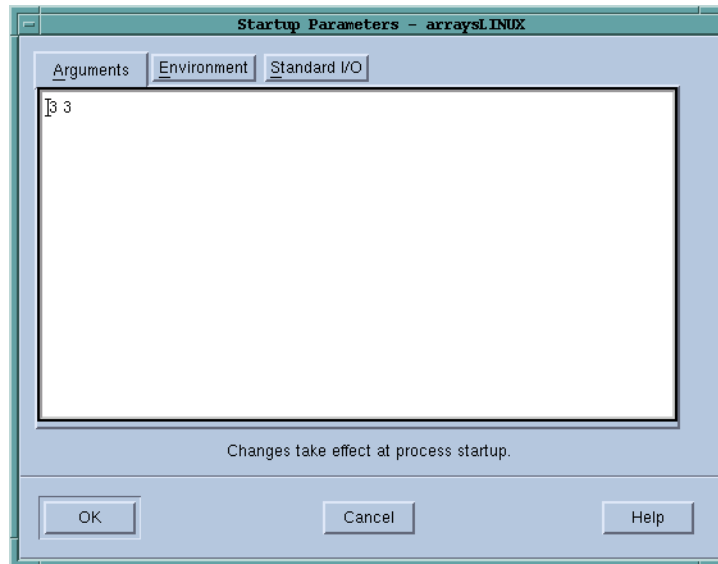
TotalView uses them arguments whenever it starts a process. The arguments are *not* global. That is, each process can have its own set of arguments, and setting arguments for one process does not effect arguments that another process uses.

You can enter arguments in two ways:

- Place them on separate lines.
- Separate them with blanks.

If either case, an argument must be entered on one line. TotalView will rewrap what you type, so do not be concerned with how it looks in this window.

Figure 40: Process > Startup Parameters Dialog Box: Showing Arguments Page



Here are some special cases:

- If an argument contains embedded blanks, enclose the argument in quotation marks (").
- If an argument contains a quotation mark, precede it with a backslash.
- If an argument contains a backslash character (\), precede it with a second backslash.
- TotalView interprets `\n` as an embedded newline.

If you delete these arguments before execution begins, TotalView does not use them.

Environment Page

Contains additional environment variables that TotalView passes to a process when it is next launched.

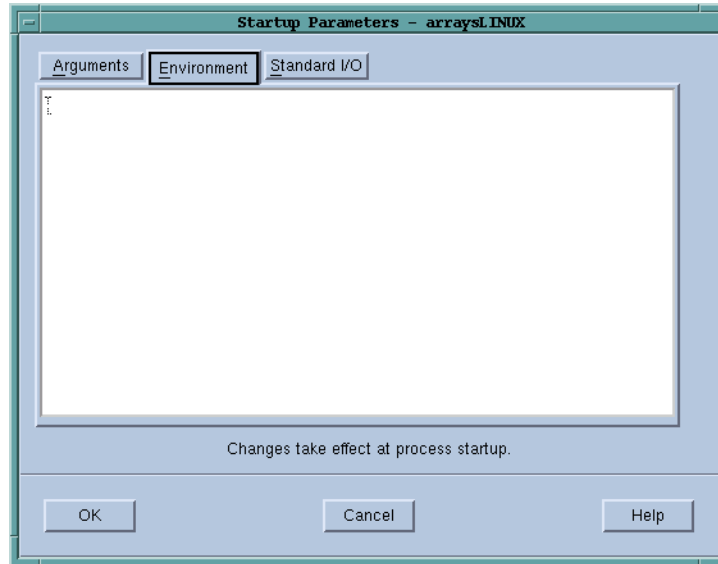
By default, a new process inherits TotalView's environment variables, and a remote process inherits **tvdsvr**'s environment variables. Using this window, you can add new variables, change the value of existing variables, or delete an existing variable.

An environment variable is specified as *name=value*. For example, the following definition creates an environment variable named **DISPLAY** whose value is **unix:0.0**:

```
DISPLAY=unix:0.0
```

Place each environment variable on a separate line.

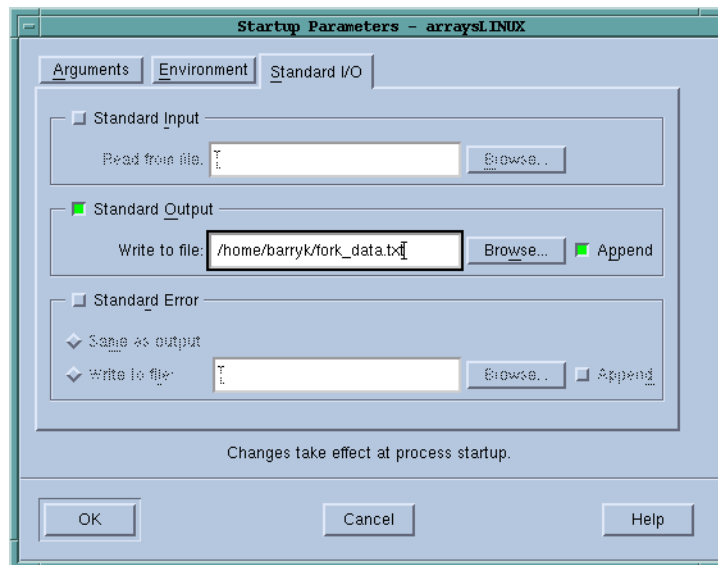
Figure 41: Process > Startup Parameters Dialog Box: Showing Environment Page



Standard I/O Page

Lets you change how TotalView handles standard input (**stdin**), standard output (**stdout**), and standard error (**stderr**). Each is handled separately. (See Figure 42 on page 89.)

Figure 42: Process > Startup Parameters Dialog Box: Showing Standard I/O Page



If you want to use the default **stdio**, **stdout**, or **stderr**, you can deselect the button that precedes the area.

Standard Input Lets you name the file that will be connected to the process's standard input (**stdin**) when it is next launched.

Processes running under TotalView's control inherit standard input from TotalView. This field lets you set the target process's standard input to be a file. You must do this before the process is created.

Read from file

If your program should receive input from a file, you can either type the file name directly or use the **Browse** button to locate the file.

Standard Output

Lets you name the file that will be connected to the process's standard output (**stdout**) when it is next launched.

Processes run under TotalView inherit their standard output from TotalView. This field lets you set the target process's standard output to a file. You must do this before the process is created.

Write to file

If you want your program to send output to a file, you can either type the file name directly or use the **Browse** button to locate the file.

stdout is buffered. If it is pointed to a file, the last few lines of the program's output are not actually written to the file until the buffer is flushed. If the target process terminates abnormally or if TotalView deletes it, the last few lines of output may never be written to the file.

Standard Error

Lets you name the file that will be connected to the process's standard error (**stderr**) when it is next launched.

Processes run under TotalView inherit **stderr** from TotalView. This field lets you set the target process's **stderr** to a file. You must do this before the process is created.

stderr is buffered. If it is pointed to a file, the last few lines of the program's output are not actually written to the file until the buffer is flushed. If the target process terminates abnormally or if TotalView deletes it, the last few lines of output may never be written to the file.

Write to file

If you want your program to send error information to a file, you can either type the file name directly or use the **Browse** button to locate the file.

Same as output

If you would like **stderr** to go to the same file as **stdout**, select this check box.

Thread Menu Commands

The following commands appear on the Thread pulldown:

- Thread > Go on page 91
- Thread > Halt on page 91
- Thread > Next on page 91
- Thread > Step on page 92
- Thread > Out on page 92
- Thread > Run To on page 93
- Thread > Next Instruction on page 93
- Thread > Step Instruction on page 94
- Thread > Set PC on page 94
- Thread > Hold on page 94
- Thread > Continuation Signal on page 94

Thread > Go

Continues just the current thread (the thread of interest) without starting other threads in the process.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Thread > Halt

Stops execution of this thread without affecting other threads in the process. After the thread stops, TotalView update this window.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Thread > Next

"Next-steps" the current thread over a source line. That is, TotalView executes one line in this thread. If the line contains a function call, TotalView executes the call as if it were a single statement. If more than one statement exists on the line, all are executed.

While this thread is being stepped, no other process executes.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Thread > Step

Steps the current thread over a source line. That is, TotalView executes one line in this thread. If the line contains a functions call, TotalView steps into the function. If more than one statement exists on the line, all are executed.

While this thread is being stepped, no other process executes.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Thread > Out

Continues the current thread until execution control returns from the function in which execution had stopped. Other threads do not run freely while this action is occurring.

You can tell TotalView to return from more than one level in the call stack by selecting the routine to which TotalView should run to in the Stack Trace Pane. For example, if routine A calls routine B and routine B calls routine C, you can to return to routine A by selecting routine A in the Stack Trace Pane. In this case, the selected routine name is the routine you want to return to.



Be careful to distinguish between this command and Run To. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Thread > Run To

Tells TotalView that it should let the thread run until it reaches the selected source line or instruction in the Source Pane. Other threads do not run freely while this action is occurring.

Contrast the action performed by this command with the process and group versions.

- The process version runs all threads in the process.
- The group version runs all processes in the control group containing the thread of interest.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

Be careful to distinguish between this command and Run To. The difference is that "Run To" requires you to select a target line. "Out" either takes you out of the current routine or to a selected target routine.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Thread > Next Instruction

"Next-steps" the current thread over one assembler instruction. That is, TotalView executes one instruction in this thread. If the instruction contains a function call, TotalView executes the call as if it were a single statement.

While this thread is being stepped, no other process executes.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see "Using Groups, Processes, and Threads" in the "TotalView Users Guide".

Thread > Step Instruction

Steps the current thread over one assembler instruction. That is, TotalView executes one instruction in this thread. If the instruction contains a function call, TotalView steps into the function.

While this thread is being stepped, no other process executes.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see “Using Groups, Processes, and Threads” in the “TotalView Users Guide”.

Thread > Set PC

Changes the program counter (PC) to the value of the selected line. That is, before you select this command, select a line in the Source Code Pane. You will be asked for confirmation before TotalView proceeds.

If the Source Pane is displaying the source for the selected stack frame and if the selected frame is not on the top of the stack, TotalView will attempt to unwind the stack and restore the registers before adjusting the PC. If it needs to perform these operations, it asks if it is OK to proceed.

Thread > Hold

When this command is checked, you are telling TotalView that it should hold the thread. When it is unchecked, the thread can run when it receives a *go* command.

Hold (checked): Holds the thread. This stops the thread and places a hold on it. *Hold* means that a *go* command cannot tell this thread that it should begin executing. Only after you release the thread is it eligible to run.

Release (unchecked): Releases the thread. *Releasing* means that the thread will run if it subsequently receives a *go* or other run-level command.



Thread stepping and asynchronous thread controls are not available when you are debugging an IRIX pthread program. These features are supported when your compiler used sprocs; for example, the MIPSpro OpenMP compiler.

For more information on processes and threads and their behavior while being stepped, see “Using Groups, Processes, and Threads” in the “TotalView Users Guide”.

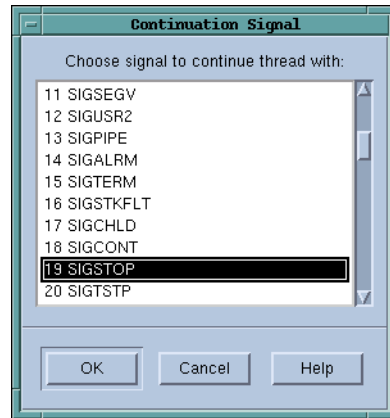
Thread > Continuation Signal

Use this dialog box to tell TotalView to send a signal to the thread of interest when the thread resumes execution.

Select the name of the signal that TotalView will send to the selected thread the next time it is continued.

If you do not want a signal sent to the thread, select the first (**No Signal Pending**) item in the list.

Figure 43: Thread > Continuation Signal Dialog Box



Selecting a signal for a thread clears the continuation signals for all other threads for the same process. In other words, only one thread can have a continuation signal set at one time.



The signal is only sent the first time the thread is continued. If you need it sent a second time, you will need to reenter this command.

Action Point Menu

The following commands are on the **Action Point** pulldown:

- Action Point > Set Breakpoint on page 95
- Action Point > Set Barrier on page 96
- Action Point > At Location on page 96
- Action Point > Enable on page 96
- Action Point > Disable on page 96
- Action Point > Delete on page 96
- Action Point > Properties on page 96
- Action Point > Suppress All on page 102
- Action Point > Delete All on page 102
- Action Point > Load All on page 102
- Action Point > Save All on page 103
- Action Point > Save As on page 103

Action Point > Set Breakpoint

Tells TotalView to set a breakpoint at the selected line or instruction. The properties of this breakpoint are:

- When Hit, Stop Group
- Enabled
- Plant in Share Group

For more information, see "Action Point > Properties" on page 96.

Action Point > Set Barrier

Tells TotalView to set a barrier breakpoint at the selected line or instruction. The properties of this breakpoint are:

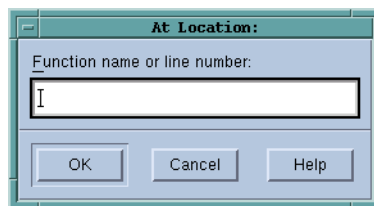
- When Hit, Stop Group
- When Done, Stop Group
- Enabled
- Plant in Share Group

For more information, see “Action Point > Properties” on page 96.

Action Point > At Location

Use this dialog box to tell TotalView to toggle an action point. That is, if an action point is not at the location, TotalView creates one. If, however, an action point exists at this location, TotalView disables it.

Figure 44: Action Point > At Location Dialog Box



- If you enter a line number, TotalView either sets or disables an action point at that line.
- If this line does not contain an executable statement, TotalView acts upon the next source line that has executable code.



You can also set action points by clicking on a line number in the Source Pane.

Action Point > Enable

Enables a previously disabled action point. That is, when execution reaches the line or instruction containing this breakpoint, TotalView will perform the action point's activity.

Action Point > Disable

Disables the selected action point. Disabling an action point leaves it set within TotalView but makes it inactive. That is, when execution reaches a disabled action point, TotalView ignores it. In contrast, if an action point is enabled, TotalView performs the action point's activity.

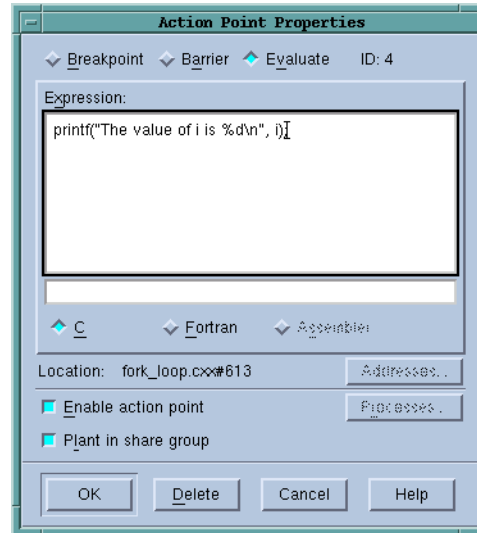
Action Point > Delete

Deletes the action point associated with the current line or instruction.

Action Point > Properties

Use this dialog box to modify the properties of an existing action point. This dialog box lets you control attributes of your action point as well as change it from one kind of action point into another. If you are creating an evaluation point, you will use this window to enter your C, Fortran, or Assembler code.

Figure 45: Action Point > Properties Dialog Box



The only kind of action point not controlled by this dialog box is the watchpoint. Use the **Tools > Create Watchpoint** command within a Variable Window when you want to set or alter a watchpoint.

The following three controls let you set or change what will happen when a program encounters an action point:

- | | |
|-------------------|--|
| Breakpoint | When an executing thread encounters a breakpoint, it stops at the breakpoint. Other controls let you indicate if the thread's process or control group will also stop. |
| Barrier | Process barrier breakpoints are similar to simple breakpoints, differing in that they let you to synchronize a group of processes in a multiprocess program. Other controls let you indicate if the thread's process or control group will also stop and what condition must be satisfied for TotalView to release threads held at the barrier. |
| Evaluate | An evaluation point is a breakpoint that has code associated with it. When a thread or process encounters an evaluation point, it executes this code. You can use evaluation points in several different ways, including as conditional breakpoints, thread-specific breakpoints, countdown breakpoints, and for patching code fragments into and out of your program. |

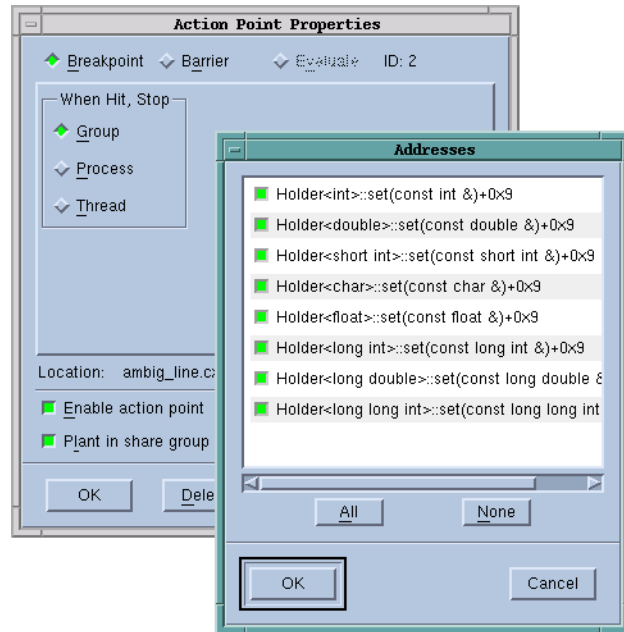
General Controls

The following four controls are used by all three kinds of action points.

Addresses

If the selected line could be mapped to more than one place—for example, you selected a line in a template or an inline function—selecting this button tells TotalView to display a dialog box that lets you refine where TotalView places breakpoints. Do this by individually check or uncheck the locations at which the breakpoint will be set.

Figure 46: Addresses Dialog Box



Process

Lets you indicate which process in a multi-process program will have enabled breakpoints. After selecting this button, TotalView displays a dialog box similar to the one it displays when you select the **Addresses** button. Note that if **Plant in share group** is selected, this button is not enabled because you've told TotalView to set the breakpoint in all processes. This dialog box

Enable action point

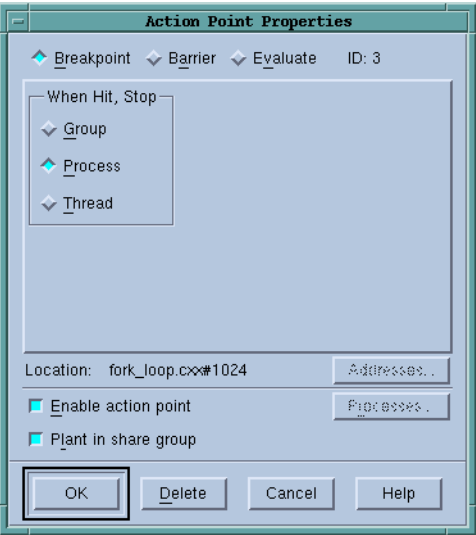
When set, TotalView will activate the action point. If this control is not set, TotalView ignores the action point.

Plant in share group

When set, the action point is shared among all of the threads in the thread's share group. In all cases, TotalView places an action point in each member of the share group. If this option is set, all of these points are active (enabled). If this option is *not set*, only this action point is enabled; the others are disabled.

Breakpoint: Setting a breakpoint tells TotalView that when execution reaches this line, it should not allow execution to continue. (See Figure 47 on page 99.)

Figure 47: Action Point > Properties Dialog Box



The **When Hit, Stop** radio buttons indicate what other threads TotalView should stop when execution reaches the breakpoint, as follows:

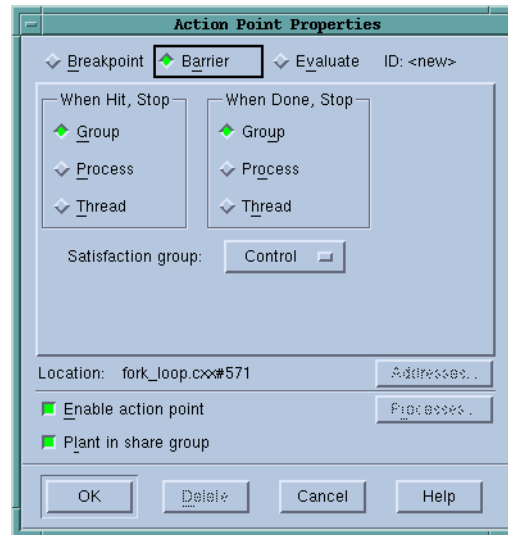
Scope	TotalView will:
Group	Stop all threads in the current thread's control group.
Process	Stop all threads in the current thread's process.
Thread	Only stop this thread.

You need to be careful when setting this attribute. For example, if you tell TotalView that it should stop the thread when the breakpoint is hit and you also tell it that the satisfaction set is, for example, the workers group, your thread will stop but the remainder of the workers group will continue executing. In most cases, this is not what you would want as most of the time you will be stopping an individual thread to examine all of the program's state at that time and TotalView cannot give you the entire state unless the other threads are also stopped.

Barrier: Select this radio button to set a process barrier breakpoint. Process barrier breakpoints are triggered when execution arrives at a line or instruction. (See Figure 48.)

Barrier breakpoints are most often used to synchronize a set of processes and threads. When a thread reaches a barrier, it stops, just as it does for a breakpoint. The difference is that TotalView prevents—that is, holds—each thread reaching the barrier from responding to resume commands (for example, *step*, *next*, or *go*) until all threads in the affected set arrive at the barrier. When all threads reach the barrier, TotalView considers the barrier to be *satisfied* and releases these threads. *They are just released; they are not*

Figure 48: Action Point > Properties Dialog Box



continued. That is, they are left stopped at the barrier. If you now continue the process, those threads stopped at the barrier also run. This is in addition to any other threads that were not participating with the barrier.

If a process is stopped and then continued, the held threads, including the ones waiting on an unsatisfied barrier, do not run. Only the unheld threads run.

When Hit, Stop Indicate what other threads TotalView should stop when execution reaches the breakpoint, as follows:

Group: Stop all threads in the current thread's control group.

Process: Stop all threads in the current thread's process.

Thread: Only stop this thread.

After all processes or threads reach the barrier, TotalView releases all held threads. (*Released* means that these threads and processes can now run.)

When Done, Stop

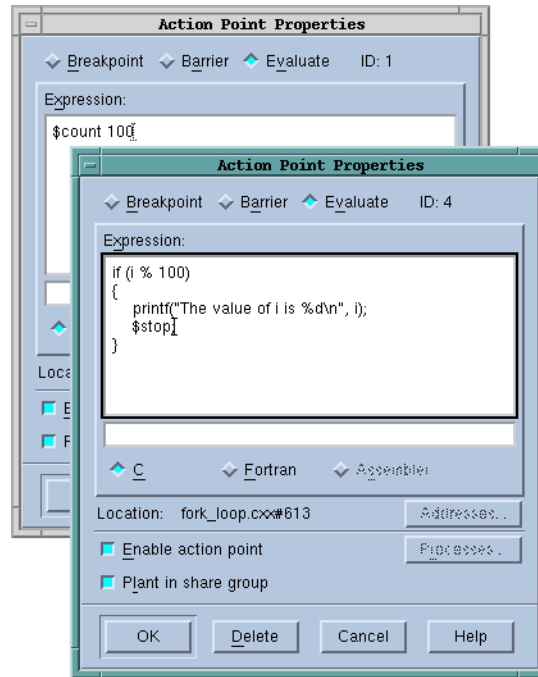
radio buttons tell TotalView what else it should stop. The meaning of these buttons is the same as the buttons in the **When Hit, Stop** area.

Satisfaction group

For even more control over what is stopped, you can indicate a *satisfaction set*. This set indicates which threads must be held before any of these held threads can be released. That is, the barrier is *satisfied* when all of the indicated threads are held. Your selection here tells TotalView that the satisfaction set consists of all threads in the current thread's **Share**, **Workers**, or **Lockstep** group.

Evaluate: When your program encounters an evaluation point, the code you enter is executed. TotalView can either compile or interpret the code you enter here. On some machines, it can only interpret your code. If TotalView can compile your code, it patches the compiled code into your process. If TotalView must interpret your code, TotalView executes this information after your program reaches the evaluation point. In either case, the code executes *before* the code within the source line executes.

Figure 49: Action Point > Properties Dialog Box



When entering code, indicate which programming language you are using. You can specify **C**, **Fortran**, or **Assembler**.

Chapter 14 of the *TotalView Users Guide* contains an extensive evaluation points discussion.

Here are some examples of ways to use evaluation points:

Countdown Breakpoints

The following is an example of a C language countdown breakpoint:

```
static int count = 100;
if (count-- == 0) {
    $stop;
    count = 100;
}
```

Conditional Breakpoints

The following is an example of a C language conditional breakpoint:

```
if (index < 0 || ptr == 0)
    $stopall;
```

Conditional Barriers

The following is an example of a C language conditional barrier point:

```
if (index < 0 || ptr == 0)
    $holdstopall;
```

In these examples, the **\$stop**, **\$stopall**, and **\$holdstopall** special directives stop a process when a condition is true. For more information, see “**Built-In Statements**” in the *TotalView Users Guide*.

Action Point > Suppress All

Toggles the state of all action points between suppressed and unsuppressed, as follows:

Suppressed—command is selected

TotalView saves the current enabled/disabled state of each action point in the process, in addition to disabling the action points.

Unsuppressed—command is not selected

TotalView restores the saved enabled/disabled state of actions points.

Suppressing and *disabling* action points do similar things. If you suppress or disable an action point, TotalView ignores it when it is encountered. They differ in that *suppressing* disables all action points. In contrast, you can only disable action points individually. In other words, *suppress* means *disable all*.

When you suppress action points, you are also telling TotalView that it can not create additional points.

Action Point > Delete All

Removes all the action points in the current Process Window.

You can delete suppressed and disabled action points.

After selecting this command, TotalView asks if this is really what you meant to do.

Action Point > Load All

Reads and sets the action point information previously written to a file. See **Action Point > Save All** on page 103 for more information.



TotalView can be told to automatically write this information to the file at the end of a session and to reload it when you start a new session. For more information, see the Action Points Page within the File > Preferences command.

When TotalView sets these action points, it will overwrite any changes you have made. For example, if you had changed a breakpoint to an evaluation point and entered code in this evaluation point, executing this command would change the evaluation point back to a breakpoint. Similarly, if you delete one of the action points saved in this file, it is restored. Reloading the saved action point does not, however, alter action points added to other lines.

**Action Point >
Save All**

Writes information about all current action points (except watchpoints) to a file. These action points are placed in a file named **program.TVD.breakpoints**. You can restore these saved action points at a later time by using the **Action Point > Save As** command.

If a file with this name already exists, TotalView overwrites it.



*TotalView can be told to automatically write this information to the file at the end of a session and to reload it when you start a new session. For more information, see the Action Points Page within the **File > Preferences** command. Also, you can reload a previously saved file by using the **Action Point > Save As** command.*

**Action Point >
Save As**

Saves all of your action point information to a file that you will name in the dialog box that will appear.



*TotalView can be told to automatically write this information to the file at the end of a session and to reload it when you start a new session. For more information, see the Action Points Page within the **File > Preferences** command.*

Use the **Action Point > Load All** command to load a saved action point file.

Tools Menu Commands

The following commands are on the **Tools** pulldown:

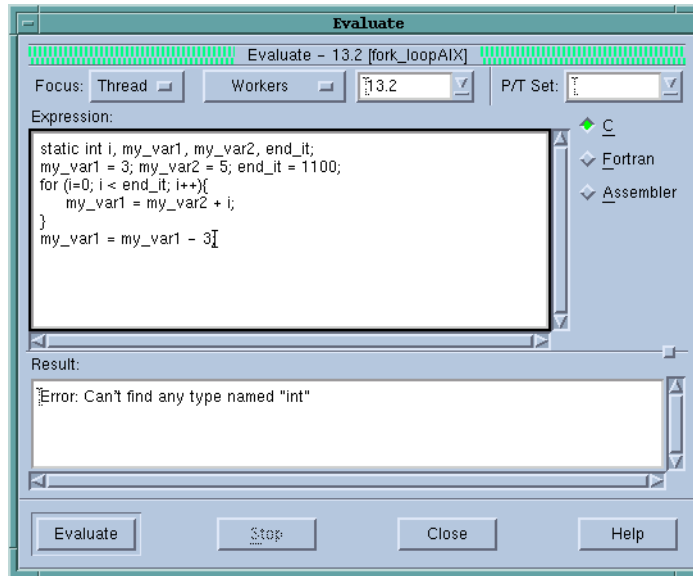
- **Tools > Evaluate** on page 103
- **Tools > Expression List** on page 105
- **Tools > Program Browser Window** on page 155
- **Tools > Fortran Modules Window** on page 149
- **Tools > Call Tree** on page 105
- **Tools > Debugger Loaded Libraries** on page 106
- **Tools > Memory Debugging** on page 107
- **Tools > Memory Block Properties** on page 107
- **Tools > Memory Event Details** on page 109
- **Tools > Thread Objects Window** on page 207
- **Tools > Message Queue Window** on page 161
- **Tools > Message Queue Graph** on page 110
- **Tools > Create Checkpoint** on page 112
- **Tools > Restart Checkpoint** on page 115
- **Tools > PVM Tasks Window** on page 169
- **Tools > Global Arrays** on page 117
- **Tools > Command Line** on page 117

Tools > Evaluate

Use this window to enter and evaluate small fragments of C, Fortran, or assembler code. This code can contain local variable declarations. In C, the data types you can use are **char**, **short**, **int**, **float**, **double**, and pointers to these data types. In Fortran, the data types you can use are **INTEGER**, **REAL**, **DOUBLE PRECISION**, and **COMPLEX**. Your code fragment can also contain references to the target process's variables. The expression can reference

local (stack) and global variables. TotalView evaluates stack variables in the context of the currently selected stack frame. (See Figure 50.)

Figure 50: Tools > Evaluate Window



Use the following controls to select processes and threads:



You'll find extensive information on the meaning of these controls in the TotalView Users Guide.

Width	Limits TotalView's selection of what executing elements may be selected. Your choices are All , Group , Process , and Thread . This control tells TotalView what the group, process, or thread of interest.
Group	Limits TotalView's selection of what executing elements within the <i>Width</i> it should chose. Your choices are Control , Share , Workers , and Lockstep . For example, if you have chosen process width and select the lockstep group, you are telling TotalView that it should select all members of the lockstep group within the current process.
P/T Selector	Tells TotalView which thread should be displayed or which thread is the thread of interest. If you need to specify more than one thread, use the P/T Set control.
P/T Set	Allows you directly enter CLI commands that create a process or thread set. For example, you could create a union of P/T sets in this control.

The remaining controls let you enter the code that TotalView will evaluate.

Expression	Enter the code to be evaluated within this text area.
C, Fortran, Assembler	The programming language in which you are writing your code.

Result	The value of the last expression in the code being evaluated.
Evaluate	Tells TotalView to evaluate your code.
Stop	Stops the evaluation of an expression.

Statements can affect variables in the target process. This means that assignment statements in an expression will change values in your program.

If a breakpoint is set within a function called from within your expression (or stops for any other reason), the expression window is *suspended*. You can debug the called routine normally as though it had been encountered during normal program operation.



You cannot use the Evaluate Window while it is suspended. To evaluate a second expression while the first one is suspended, just open a second Evaluate window for the process.

For more information, see the "Evaluating Expressions" section within the "Setting Action Points" chapter of the *TotalView Users Guide*.

Tools > Expression List

For information, see "Expression List Window" on page 219.

Tools > Program Browser

For information, see **Program Browser Window** on page 155.

Tools > Fortran Modules

For information, see **Fortran Modules Window** on page 149.

Tools > Call Tree

Displays a window that will show a graphical representation of your program's stack. This representation, which is called a *call tree*, is a dynamic representation of your program's state.

Functions and subroutines are displayed as boxes. The lines linking functions and subroutines to one another indicate that one routine was called by the other, with the arrow pointing to the called routine.

The number next to a link indicates the number of times one routine is currently called by another in all threads in the selected scope.

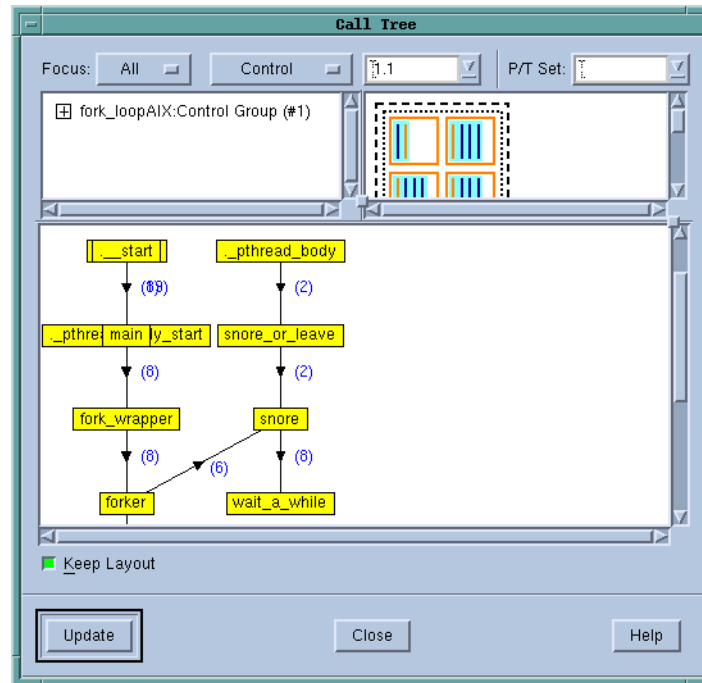
Use the following controls to select processes and threads:



You'll find extensive information on the meaning of these controls in the TotalView Users Guide.

Width	Limits TotalView's selection of what executing elements may be selected. Your choices are All , Group , Process , and Thread . This control tells TotalView what the group, process, or thread of interest.
Group	Limits TotalView's selection of what executing elements within the <i>Width</i> it should chose. Your choices are Control , Share , Workers , and Lockstep . For example, if

Figure 51: Tools > Call Tree Window



you have chosen process width and select the lockstep group, you are telling TotalView that it should select all members of the lockstep group within the current process.

P/T Selector

Tells TotalView which thread should be displayed or which thread is the thread of interest. If you need to specify more than one thread, use the **P/T Set** control.

P/T Set

Allows you directly enter CLI commands that create a process or thread set. For example, you could create a union of P/T sets in this control.

Use the **Update** control to tell TotalView to update the display. You would do this if you have asked TotalView to display a call tree while your program is executing or if you have altered the P/T set that TotalView will display.

Changing Positions: After TotalView displays the graph, you can rearrange the way this information is displayed by dragging the rectangles to new positions.

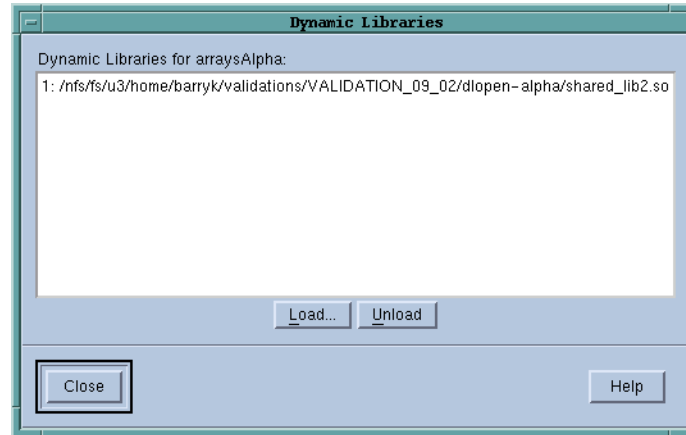
If you select the **Keep Layout** check box, TotalView remembers the changes you make when you select the **Update** button. If it isn't selected, TotalView discards your changes when it updates the graph.

Tools > Debugger Loaded Libraries

Displays a dialog box into that lets you tell TotalView that it should immediately open a shared library using **dlopen()**.

As your program executes, TotalView loads in shared libraries. In some cases, you may want to bring them into memory before they are needed. For example, you may need to refer to a function contained within the

Figure 52: Tools > Debugger
Loaded Libraries Window



library when you are creating an evaluation point or using the **Tools > Evaluate** window.

After a library is brought into memory, you can refer to any symbol contained within the library. After TotalView loads the library, it asks if you want to set breakpoints in the library.

The buttons that are unique to this dialog box are:

- | | |
|---------------|---|
| Load | Tells TotalView to display a dialog box that you can use to locate the library within your file system. |
| Unload | Deletes the selected library. |

When you are done, select the **Close** button. As the libraries are already loaded, selecting this button just dismisses the dialog box.

Tools > Memory Debugging

Tools > Memory Block Properties

For information, see **Memory Debugging Window** on page 175.

The **Memory Block Properties** Window displays information about all of the blocks that you asked the Memory Debugger to keep track of by using the **Tools > Block Properties** command. You can display this window in two different ways. Pressing the **Hide Backtrace Information** conceals most of what is displayed in Figure 53 on page 108.

After pressing this button, the window reconfigures itself to that shown in Figure 54 on page 108. When this window is being displayed, pressing **Show Backtrace Information** shifts it back.

As an alternative, you can both make the window larger and use the splitter control to enlarge the top area.

The information within the bottom area is essentially the same as that which is displayed when you generate a Backtrace View in the Memory Debugger. You will find information on the contents of the Block Backtrace Information area within "**Heap Status Page**" on page 191

The first line contains a graphical summary of the blocks information. It contains:

Figure 53: Tools > Memory Block Properties Window

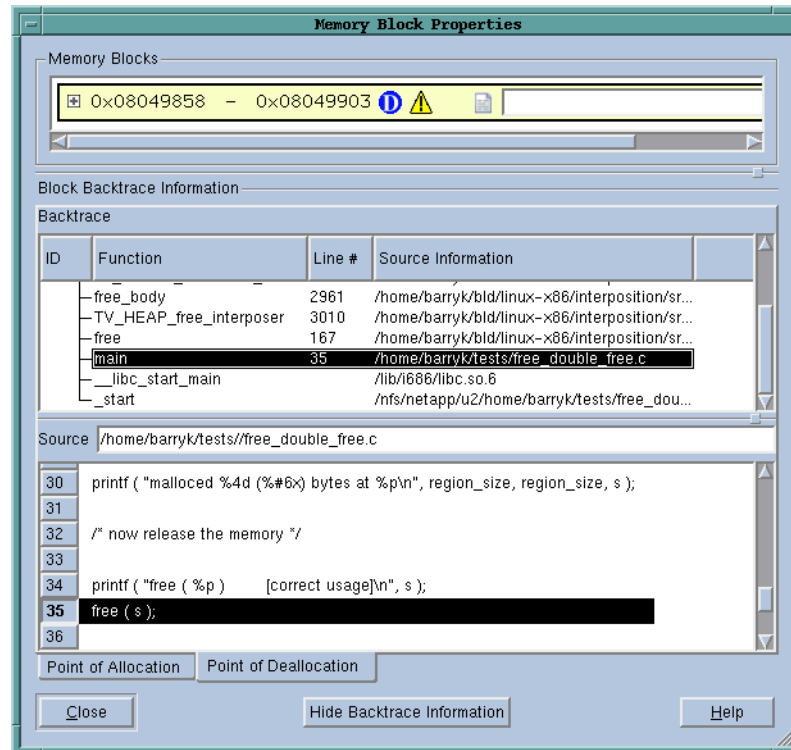
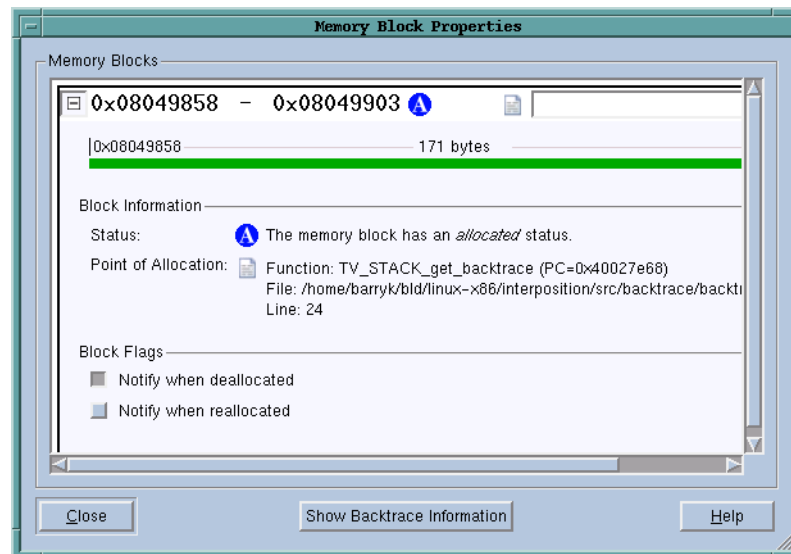



Figure 54: Tools > Memory Block Properties Window



- **Memory block extent:** the numbers in bold indicate the starting and ending address of the block. For example, the block in Figure 54 indicates the block starts at 0x08049858 and ends at 0x08049903.
- **Status:** an indicator indicating if the block is allocated, deallocated, leaked, or being hoarded. For example, the **A** in Figure 54 indicates that the block is allocated.

Tools > Memory Event Details

- **Point of Allocation:** if you place your mouse over the  icon, TotalView displays a tool tip that shows the function in which the block was allocated, the function's source file, and the line number within that file.
- **Comment:** if you have more than one or two memory blocks, you might want to enter a comment in the text box to remind yourself what the block is.

The next lines within the Memory Blocks area restate the information that was presented in the summary area. The color of the block's graphic is the same as that used in the Heap Status Page's Graphical View.

The Block Flags area contains two check boxes:

- Notify when deallocated
- Notify when reallocated

Checking a box tells the Memory Debugger that it should stop execution when the block is deallocated or reallocated and display the Memory Event Details Window. This allows you to track when your program is managing memory and what is being managed. For example, if you have a double free problem, obtaining notification lets you know where the block is first freed.

When a memory event occurs, the Memory Debugger automatically displays this window. After you dismiss it, you can redisplay it using this command.

This window has four areas, as follows:

- The top line tells you what type of error or event occurred.
- The **Block Information** area gives the memory location of the block and its status.
- The third area contains the function backtrace if the error or event is related to a block allocated on the heap. The Memory Debugger retains information about the backtrace that existed when the memory block was allocated and the backtrace when it was deallocated. You can tell the Memory Debugger which it should display by selecting either the **Point of Allocation** or **Point of Deallocation** tab.

If a memory error occurred, the deallocation backtrace is often the same as the backtrace being shown in the Process Window's Source Pane. If the memory error occurs after your program deallocated this memory, the backtraces are different.

- The bottom area shows you where the allocation or deallocation occurred in your program.



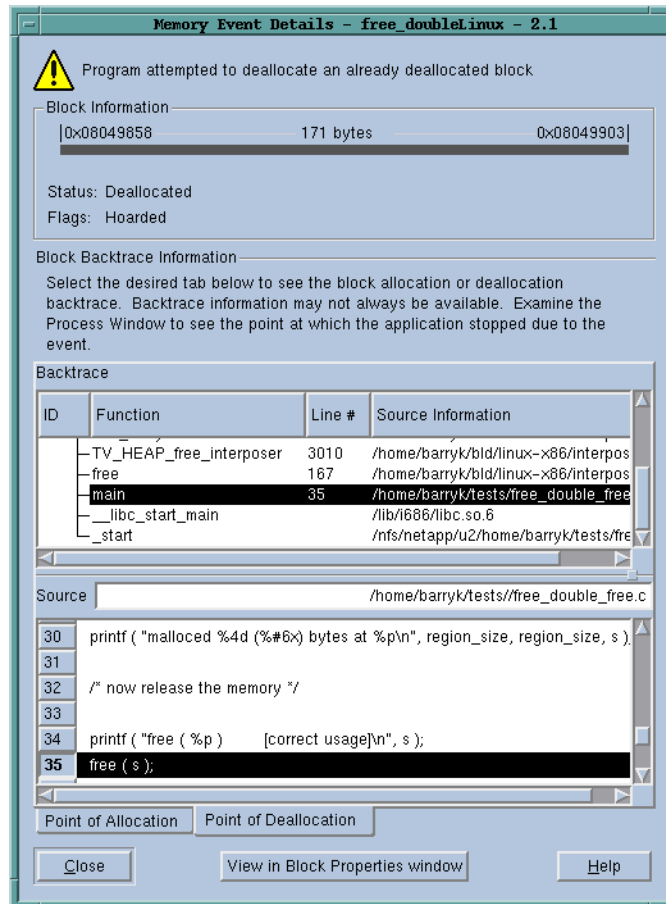
In some cases, the Memory Debugger does not display an allocation backtrace. For example, if you try to free memory allocated on the stack or in a data section, there's no backtrace because your program did not allocate the memory.

If you need to redisplay the Memory Block Window after you dismiss it, select the **Tools > Memory Event Details** command.

Tools > Thread Objects

For more information, see **Thread Objects Window** on page 207.

Figure 55: Tools > Memory Event Details Window



Tools > Message Queue

For information, see **Message Queue Window** on page 161.

Tools > Message Queue Graph

Displays a window that shows a graphic representation of the state of message queue information. (The **Tools > Message Queue** command tells TotalView to display this information in a non-graphical form.)

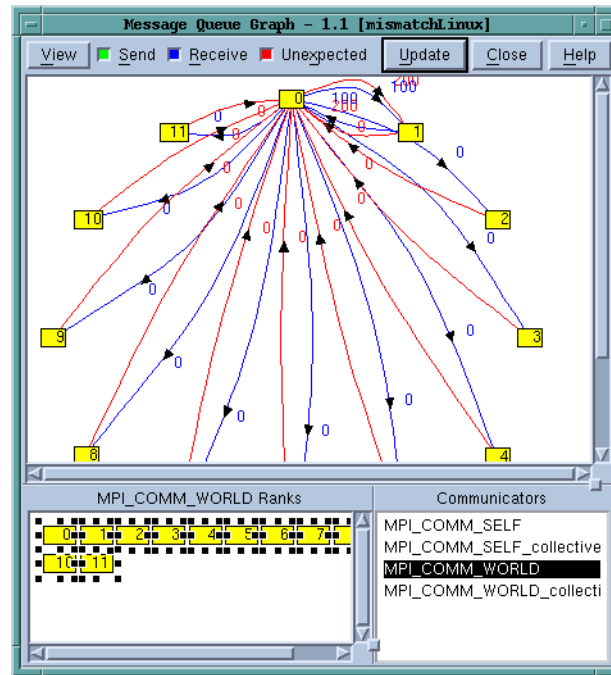
This window has a four-item context menu:

- **Dive:** dives into a process. See **Diving** on page 112 for more information.
- **Attach:** If you've detached from a rank, you can use this command to re-attach to it.
- **Detach:** Removes the selected rank from the display.
- **Subset Attach:** displays the same dialog box as is displayed when you select the **Group > Attach Subset** on page 72 dialog box. It differs in that it knows the current context.

This visual display of message queue information is often used to spot unexpected messages and to uncover deadlocks. Here is a procedure for displaying this information:

- 1 Select one or more message types to be displayed. By default, TotalView displays **Unexpected** messages. However, you can display any combination of **Send**, **Receive**, and **Unexpected** messages.

Figure 56: Tools > Message Queue Graph Window



- 2 Select the ranks for which you want information. The ranks are displayed in the pane in the lower left corner of the window. Ranks to which TotalView is attached are displayed as yellow numbered buttons. If a button isn't numbered, TotalView is not attached to the rank.
Use your mouse to select processes. You can either select processes individually by left-clicking on the rank number or you can use your mouse to draw a rectangle around a group of rank numbers. If you want to remove an already selected rank, click on it while pressing the Control key.
- 3 Select the communicators to be included in the display; for example, **MPI_COMM_WORLD**.
- 4 Press the **Update** button.

TotalView responds by displaying the message queue graph. Here, ranks are drawn as yellow boxes. The number within a box is the process's rank.



Wildcards are represented by a made-up process whose name is "ANY". For example, a request such as "receive message with tag 400 from any source" means that TotalView will create a process labeled "ANY".

The messages are indicated as curved lines with an arrow at the end. This arrow indicates the rank receiving the message. The message's tag (or number) is displayed near the line. The line's color represents the kind of message:

- **Red**: unexpected messages
- **Blue**: pending receives
- **Green**: pending sends; these messages seldom occur

TotalView only shows MPI message queue information for processes that are halted or at a breakpoint at the time when you selected the **Update**

button. That is, if some of your MPI processes are running when you select **Update**, the message queue display will be inaccurate and incomplete.

Diving: You can dive into a process or a link by double-clicking on it.

- Double-clicking on a process tells TotalView to display a Process Window.
- Double-clicking on a link displays the window that is displayed when you select the **Tools > Message Queue Window** command.

Changing Graph Item Positioning: After TotalView displays the graph, you can rearrange the way this information is displayed by changing:

- Where processes are displayed by dragging the yellow process rectangles to a new position.
- The curve of the lines linking processes by clicking on the line and dragging the displayed rectangular dragging handles. If only one rectangle is visible, then one rectangle is on top of the other.

If you select the **Keep Layout** check box, TotalView remembers the changes you make when you select the **Update** button. If it isn't selected, TotalView discards your changes when it updates the graph.

Performance Considerations: The procedure used to extract MPI information from your program can be slow. Consequently, TotalView may take a considerable time responding to your request. (The time TotalView takes is related to the number of processes being polled for information.)

Tools > Create Checkpoint

This command is only available on SGI IRIX and IBM RS/6000.

Saves a program and its process's information into the **Name** file.

You can then use the **Tools > Restart Checkpoint** command to restart the program at a later time. This saved information includes process and group IDs. While breakpoint information is not saved as part of the checkpoint, you can use the **Action Point > Save All** command to save the current breakpoints for later use.

You can checkpoint programs running on:

- RS/6000
- SGI IRIX

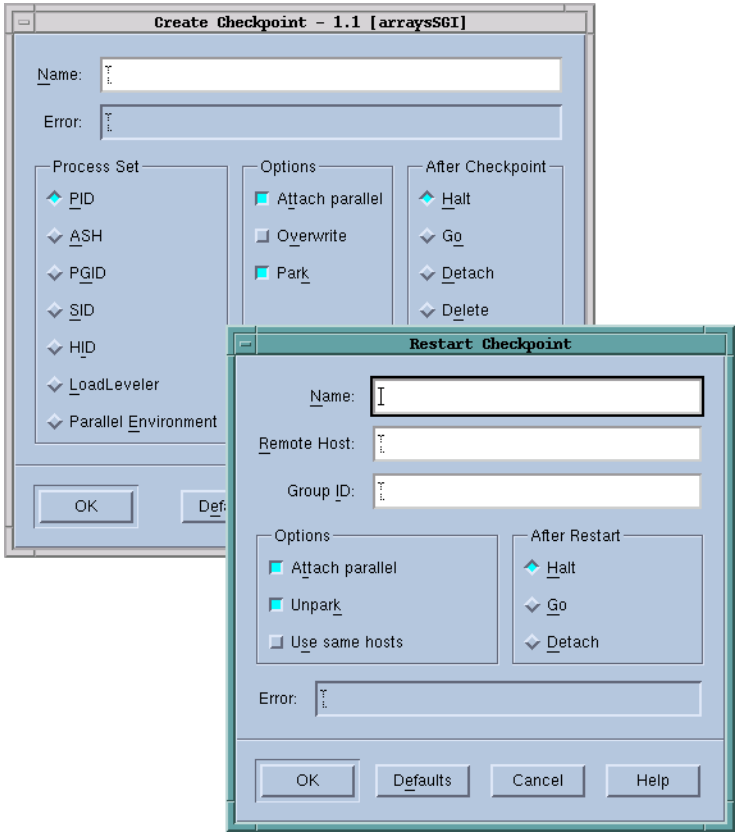
Processes running remotely that communicate by using sockets can have difficulties when being checkpointed because IRIX will not checkpoint programs with open sockets. As the TotalView Server (**tvdsrv**) uses sockets to redirect **stdin**, **stdout**, and **stderr**, you will need to use the **Standard I/O Page** of the **Process > Startup Parameters** dialog box to modify the way your processes send information to a **tty** before creating a checkpoint.



*If you are using SGI MPI, you need to use the **-cpr** command-line option, which, if conditions are correct, you'll be able to create a checkpoint. Use the ASH option with MPI checkpoints*

This command is the same as the CLI **dcheckpoint** command.

Figure 57: Checkpoint and Restart Dialog Boxes



RS/6000 If you are creating a checkpoint on an RS/6000 machine, press one of the following radio buttons to define the state of the process both before and after the checkpoint:

- Halt** Processes stop executing after they are checkpointed. This is the default option.
- Delete** Processes exit after being checkpointed.
- Error** If a problem occurs, TotalView displays the reason in this area.

The name used for the checkpoint file is set using an environment variable. See your POE documentation for more information.

SGI IRIX The controls in this window are as follows:

- Name** The name being assigned to the checkpoint. If this name already exists, TotalView overwrites it.
- Error** If a problem occurs, TotalView displays the reason in this area.
- Process Set** Indicates the set of processes that will be checkpointed. Your options are:
 - PID** Checkpoint the program indicated by a PID.
 - ASH** Checkpoint the array session.
 - PGID** Checkpoint the entire process group.

SID	Checkpoint the entire process session.
HID	Checkpoint the hierarchy rooted in the focus process.
Options	Indicates control options that you may find useful. Choose as many as needed to define your checkpoint.
Attach parallel	Tells TotalView that it should reattach to processes from which the checkpointing processes detached. (Some systems automatically detach you from processes being checkpointed.)
Overwrite	Lets TotalView assign new IDs when it restarts a checkpoint. If you do not use this option, the same IDs are used.
Park	Tells TotalView that it should hold all processes before it begins checkpointing them. If you will be restarting a checkpoint from the shell, you should make sure that this option is <i>not</i> selected. In addition, if this option is not selected and you will be restoring the checkpoint from within TotalView, you will need to select the Tools > Restart Checkpoint Unpark check box.
After Checkpoint	Defines the state of the process both before and after the checkpoint. Use one of the following options:
Halt	Processes stop executing after they are checkpointed. This is the default option.
Go	Processes continue running after being checkpointed.
Detach	Processes continue running after being checkpointed. In addition, TotalView detaches from them.
Delete	Processes exit after being checkpointed.
Defaults	Restores selections in this dialog box to their default values. These are as follows: PID , Attach parallel , and Halt .

The **Process Set** options tell TotalView which processes it should checkpoint. While the focus set can only contain one process, processes within the same process group, process session, process hierarchy, or array session can also be included within a checkpoint.

The **After Checkpoint** options let you specify what happens after the checkpoint operation concludes. The default operation is that TotalView tells the checkpointed processes that they should **Halt**, which allows you to investigate a program's state at the checkpointed position. In contrast, **Go** tells TotalView that it should let the processes continue to run. The **Detach** and

Delete options are less frequently used. **Detach** allows you to shut down TotalView and leave the processes running. The **Delete** option differs from **Detach** in that processes started by TotalView are also terminated.

Just before TotalView begins checkpointing your program, it temporarily stops (that is, *parks*) the processes being checkpointed. Parking ensures that the processes do not run freely after a **Tools > Create Checkpoint** or **Tools > Restart Checkpoint** operation. (If they did, your code would begin running before you get control of it.) If you will be restarting the checkpoint file outside of TotalView, you must deselect the **Park** check box.

TotalView detaches from processes before they are checkpointed. By default, TotalView automatically reattaches to them. If you want something different to occur, you can tell TotalView that it should never reattach by deselecting the **Attach parallel** check box.

The CLI's **dcheckpoint** command performs the same operations as this command.

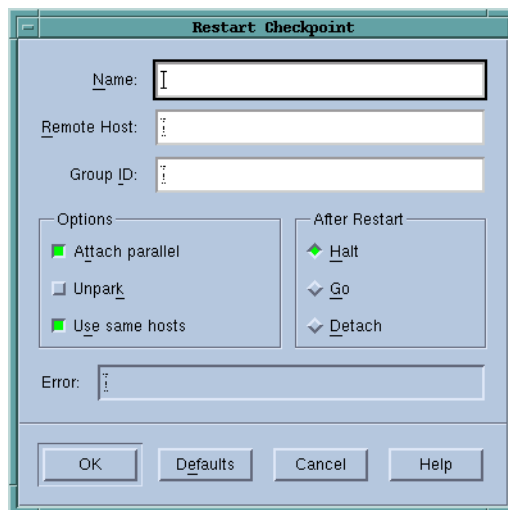
Tools > Restart Checkpoint

This command is only available on SGI IRIX and IBM RS/6000.

Use this dialog box to restore and restart all of the checkpointed processes. By default, TotalView attaches to the base process. If parallel processes are related to this base process, TotalView attaches to them. If you do not want TotalView to automatically attach to them, deselect the **Attach parallel** option.

If an error occurs while attempting to restart the program from the checkpoint, information is displayed in the **Error** area.

Figure 58: Tools > Restart Checkpoint



The CLI's **drestart** command performs the same operations as this command.

Name	Names a previously saved checkpoint file.
Remote Host	Names the remote host upon which the restart will occur.

Group ID	Names the control group into which TotalView places all created processes
Options	Indicates control options that you may find useful. If this is an RS/6000 checkpoint, Attach parallel is automatically checked and it cannot be unchecked. These options have the following meaning:
Attach parallel	If selected, TotalView attaches to parallel processes as they are being created. If this item is not selected, TotalView only attaches to the base process.
Unpark	(SGI only) Select this checkbox if the checkpoint was created outside of TotalView or if you did not select the Park checkbox within the Tools > Restart Checkpoint dialog box when you created the checkpoint file.
Use Same Hosts	(IBM only) If selected, the restart operation tries to use the same hosts as were used when the checkpoint was created. If TotalView cannot use the same hosts, the checkpoint operation fails.
After Restart	Defines the state of the process both before and after the checkpoint. You can use one of the following options:
Halt	Parallel processes are held immediately after the place where the checkpoint occurred. TotalView attaches to these created parallel processes. (This is the default.)
Go	(SGI only) Checkpointed parallel processes are started and TotalView attaches to the created processes.
Detach	(SGI only) Checkpointed process are started. TotalView does not attempt to attach to them.

Restarting on AIX using LoadLeveler: On the RS/6000, if you wish to debug a **LoadLever poe** job from the point at which the checkpoint was made, you must resubmit the program as a **LoadLeveler** job to restart the checkpoint. You will also need to set the **MP_POE_RESTART_SLEEP** environment variable to an appropriate number of seconds. After you restart **poe**, start TotalView and attach to **poe**.



*When attaching to **poe**, parallel tasks will not yet be created, so do not try to attach to any of them. Also, you'll need to set the **Attach to none** option with the **Parallel** Page of the **File > Preferences** Dialog Box.*

When doing this, you cannot use the restart the checkpoint using this command. **poe** will tell TotalView when it is time to attach to the parallel task so that it can complete the restart.

Tools > PVM Tasks

For more information, see **PVM Tasks Window** on page 169.

Tools > Global Arrays

Tools > Global Arrays command; Process window: Tools > Global Arrays; Global Arrays command Opens a window containing a list of your program's global arrays. For each global array, TotalView displays four entries:

Handle	A value assigned to the array by the Global Arrays software.
Ghosts	Indicates if Global Arrays will write ghost cells from one process to another.
C type	Defines how the array is defined in the C programming language
Fortran Type	Defines how the array is defined in the Fortran programming language.

You can see an array's data by diving on either the Fortran or C type line. After you dive, TotalView displays a standard variable window containing this information. Because it is a standard window, you can manipulate its contents using standard TotalView commands. For example, you can filter and slice the array, obtain statistics, visualize its data, and so on.

If you want to change the way TotalView displays the data from its C definition into its Fortran definition or vice versa, you can cast it, as follows:

<ga>	Casts the array so it is showing data as it would be displayed within a C program.
<GA>	Casts the array so it is showing data as it would be displayed within a Fortran program.
<Ga>	Casts the language within the current context.

Tools > Command Line

Opens the CLI window. This window is an **xterm** window into which you enter CLI commands.

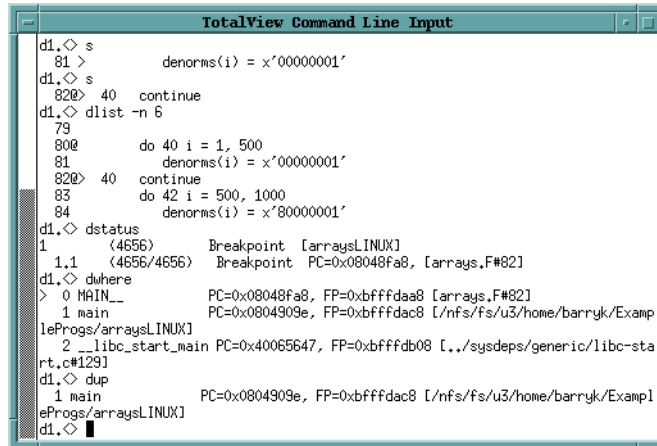
You can find information on the CLI in the *TotalView Users Guide* or by selecting **CLI** from the Help command on this window.

Window Menu Commands

The following commands are on the Window pulldown:

- **Window > Update** on page 118
- **Window > Update All** on page 118
- **Window > Duplicate** on page 118
- **Window > Memorize** on page 118
- **Window > Memorize all** on page 118
- **Window > Root** on page 119

Figure 59: Tools > Command Line (CLI) Window



```

TotalView Command Line Input
d1,< s
81 >          denorms(i) = x'00000001'
d1,< s
820> 40  continue
d1,< dlist -n 6
79
800      do 40 i = 1, 500
81          denorms(i) = x'00000001'
820> 40  continue
83      do 42 i = 500, 1000
84          denorms(i) = x'80000001'
d1,< dstatus
1      (4656)      Breakpoint [arraysLINUX]
1.1    (4656/4656) Breakpoint PC=0x08048fa8, [arrays.F#82]
d1,< dwhere
> 0 MAIN__
1 main__          PC=0x08048fa8, FP=0xbfffdac8 [arrays.F#82]
leProgs/arraysLINUX]
2 __libc_start_main PC=0x40065647, FP=0xbffdb08 [.../sysdeps/generic/libc-start.c#129]
d1,< dup
1 main__          PC=0x0804909e, FP=0xbfffdac8 [/nfs/fs/u3/home/barryk/ExampleProgs/arraysLINUX]
d1,<

```

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Duplicate

Tells TotalView that it should create a second copy of the current Process Window.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Displays (uncovers) the Root Window.

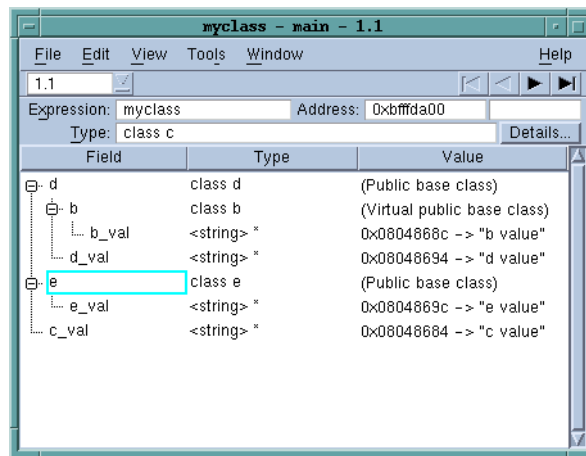
The Variable Window

3

Variable Window Overview

The Variable Window displays information about one of your program's objects.

Figure 60: Variable Window



This window is more than just a viewer. It lets you change a variable or element's value, as well as allowing you a variety of ways to alter what TotalView is displaying. For example, you can cast a variable to the way you want it to appear.



TotalView's type mapping procedure lets you control how TotalView displays information. For more information, see the TotalView Users Guide. The Creating Type Transformation chapter in the Reference Guide contains information on how you can create your own transformations.

Other examples of changes you can make are:

- If TotalView is displaying an array, you can display a limited section of the array, called a *slice*, instead of the whole array.
- You can also enter a *filter* that tells TotalView to make decisions about which array elements it should display.

Topics in this section are:

- "Expression Field—Altering What is Being Displayed" on page 122
- "Address Field—Changing a Variable's Address" on page 122
- "Status Field—Seeing State and Error Messages" on page 123
- "Type Field—Changing a Variable or Element's Data Type" on page 123
- "Slice Field—Altering Subscripts" on page 123
- "Filter Field—Filtering Array Values" on page 124

Other operations you can perform are described in the following sections.

- "Diving into Elements" on page 125
- "Sorting Columns" on page 125
- "Changing the Process and Thread" on page 125

When you tell TotalView to open a Variable Window, TotalView tries to find a window that is already displaying the element. If it finds one, TotalView moves this window to the front of the display.

If the variable or element being displayed contains substructures, only the name of the substructure is immediately visible. To display a substructure, click on the "+" icon. Clicking on the "-" icon closes the substructure, hiding the information. You can use the **View > Expand all** and **View > Collapse All** commands to expand and collapse all trees.

Expression Field—Altering What is Being Displayed: When TotalView first displays a Variable Window, the value of the **Expression** field contains a variable, element, or expression. The exact contents depend upon what you dove on or what was entered in a **View > Lookup Variable** command.

By changing this field's text, you can change what is displayed. For example, if the expression contained the variable **my_var**, you could change this to **my_var.struct1.substruct1[3]**.

As the name of this field implies, you can type an expression in this field. For example, you could type either **i+3** or **my_var[i+3]**. You cannot, however, enter an expression that contains a function call or an expression that has a side-effect. For example, you could not enter **my_var[i++]**.

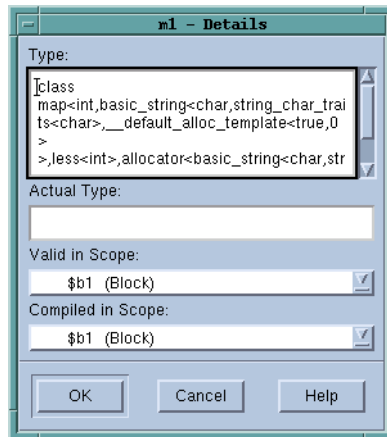
Address Field—Changing a Variable's Address: To view a different part of memory, edit the **Address** field in the upper left corner of the window. If TotalView needs to display information about an address, it displays this information in the unlabeled field to the right of the **Address** field. Note, however, that you cannot change the address of register variables.

Type Field—Changing a Variable or Element’s Data Type: To change the data type that TotalView uses to format the variable, edit the text within the **Type** field.

While you cannot directly edit the type entry for a field in a substructure, you can dive into the substructure and then edit it.

If you select the **Details** button, TotalView displays a window containing more information about the variable.

Figure 61: Details Window



For more information, see “*Variable Details Window*” on page 143.

Status Field—Seeing State and Error Messages: If a problem occurs when you make a change to a field or if TotalView needs to tell you something about this window’s contents, it will display information here.

For example, if a variable is no longer in scope—that is, your program is no longer executing in a scope in which the variable is defined—TotalView tells you that the variable is stale.

Slice Field—Altering Subscripts: To change an array’s subscripts, edit the data that appears after the **Slice:** field at the top of the window. You can enter each array dimension by using one of the following formats, depending upon your programming language:

- (slice descriptor, slice descriptor, ...)
- [slice descriptor][slice descriptor]...

The slice descriptor tells TotalView that it should display every *stride*th subscript from the lower bound to the upper bound, inclusive. If the stride is negative, TotalView shows every *–stride*th subscript from the upper bound to the lower bound, inclusive, in reverse order.

Each **Slice** field has a lower bound, an upper bound, and a stride. These elements are separated by a colon (:).

If you omit the stride value and its colon separator, the default stride value is 1. If you omit the upper bounds, the lower bounds value will be the

upper bounds value. The lower bound must be less than or equal to the upper bound, and the stride cannot be 0.

Filter Field—Filtering Array Values: You can tell TotalView that it should selectively display information from the array by entering a value in the **Filter** field.

While there are a number of ways to specify a filter, the general format is:

operator value

where *operator* is one or more of the following:

*<, <=, >, >=, ==, !=
.lt. .le. .gt. .ge. .eq. .ne.*

value can be a constant integer or real value. For example:

> 0

You can also enter a TotalView intrinsic, which is a built-in representation of IEEE floating-point NaN (Not a Number), INF (infinity), or denormalized value. These intrinsics are:

Intrinsic	Meaning
\$nan	any NaN (Not a Number)
\$nanq	quiet NaN (Not a Number)
\$nans	signaling NaN (Not a Number)
\$inf	any INF, either positive or negative
\$ninf	negative INF
\$pinf	positive INF
\$denorm	denormalized number, either positive or negative
\$pdenorm	positive denormalized number
\$ndenorm	negative denormalized number



You can only use the == and != operators with these intrinsics. For example, != \$denorm.

You can add a second component to the filter to indicate that TotalView should show elements contained within a range by using the following format:

[>|low_value:<|high_value

Here are some points to consider:

- *low_value* and *high_value* are constant integers or real values.
- You can apply the < and > operators to the low and high values to allow for exclusive ranges. By default, the range is inclusive of the lower and upper values.
- *low_value* and *high_value* cannot be of different types. For example, the following range is invalid:

*1:2000u
1.0:2000*

- You can use the `$value` token to represent the current array element. For example:

```
$value > 0 && $value < 100
```

This filter expression tells TotalView to displays all array elements that are greater than 0 and less than 100.

- Your filter can also contain program variables. For example:

```
$value != x && $value < y
```

- You cannot use function calls in a filter expression.

For more information, see Setting Action Points in the *TotalView Users Guide*.

Changing the Value of a Variable: To change the value of a simple variable or change the value of a field in a more complex variable (a structure or array), click the cursor within the variable or element's value, then enter the changed value.

Diving into Elements: To view the target of a pointer or to view one element in an array or structure, double-click on the entry. TotalView replaces the contents of the window with a view of the item you dived into (or the object it points to, in the case of pointers). This allows you to easily chase chains of linked structures.

After you dive into a field, you can edit its contents. (Diving into a substructure is the only way to edit the substructure's value.) You can also cast individual entries in arrays to different data types for display.

Sorting Columns: If you are viewing an array, you can sort the array's value by clicking on the Value heading.

- Clicking once on **Value** sorts the array into ascending order.
- Clicking again sorts the array into descending order.
- Clicking a third time returns the order to what it was before you clicked on the **Value** heading.

Changing the Process and Thread: The process/thread box initially indicates the variable or element's context when you created the Variable Window. You can change this context be either typing in a new process/thread number (be sure to separate the process number from the thread number with a period) or selecting a value from the pulldown list.

File Menu Commands

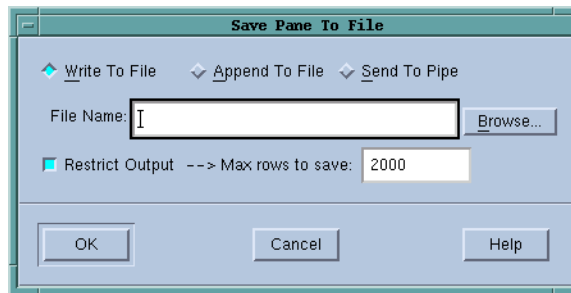
The following commands are on the **File** pulldown:

- **File > Save Pane** on page 126
- **File > Close Similar** on page 126
- **File > Close** on page 127
- **File > Exit** on page 127

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

Figure 62: File > Save Pane Dialog Box



Write to File

Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File

Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist, TotalView creates the file before writing this information.

Send To Pipe

Sends the data to the program or script named in the **File Name** field.

Restrict Output --> Max rows to save

If checked, TotalView will limit how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

File > Close Similar

Closes all Variable Windows and windows derived from this window.

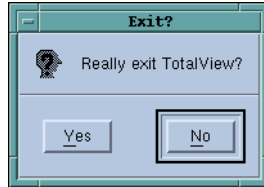
File > Close

Tells TotalView to close the current Variable Window.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

Figure 63: File > Exit Dialog Box



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The following commands are on the **Edit** pulldown:

- **Edit > Undo** on page 127
- **Edit > Reset Defaults** on page 127
- **Edit > Cut** on page 127
- **Edit > Copy** on page 128
- **Edit > Paste** on page 128
- **Edit > Delete** on page 128
- **Edit > Find** on page 128
- **Edit > Find Again** on page 129

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Reset Defaults

Selecting this command tells TotalView to undo the changes you've made to the Variable Window that don't affect information stored in memory. For example, this command changes you made to the **Type**, **Expression**, **Slice**, and **Filter** fields. Changes you make to a variable or element's value are not reset.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language state-

ment contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

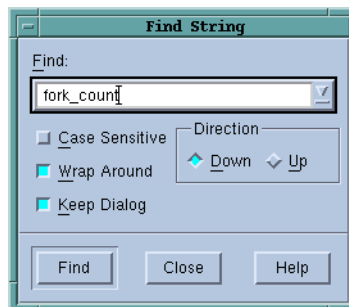
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 64: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for |

	"foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box.
<i>Direction</i>	Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the Find box.
Close	Closes the Find dialog box.

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The following commands are on the **View** pulldown:

- **View > Dive** on page 129
- **View > Dive in New Window** on page 130
- **View > Dive In All** on page 130
- **View > Expand All** on page 130
- **View > Collapse All** on page 130
- **View > Add to Expression List** on page 130
- **View > Undive** on page 130
- **View > Undive All** on page 131
- **View > Redive** on page 131
- **View > Redive All** on page 131
- **View > Laminate > None** on page 131
- **View > Laminate > Process** on page 131
- **View > Laminate > Thread** on page 131
- **View > Compilation Scope > Fixed** on page 132
- **View > Compilation Scope > Floating** on page 132
- **View > Loader Symbols** on page 132
- **View > Padding** on page 132

View > Dive

Replaces the contents of your Variable Window with information about the selected data element. TotalView lets you dive on array elements, pointers, structures, and substructure elements.

You can use the **View > Undive** command or the left-facing triangle in the window's upper right-hand corner to restore the window to what it was before you dived into an element.

View > Dive in New Window

Opens a new variable containing information about the selected element. TotalView lets you dive on array elements, pointers, structure, and sub-structure elements.

This command differs from a dive command in that TotalView displays information about the selected element in a new window.

View > Dive In All

The **View > Dive In All** command (which is also available when you right click on a field) allows you to display an element within an array of structures as if it were a simple array. After you select this command, TotalView replace the information in the current Variable Window with new information.

For example, suppose you have the following Fortran definition:

```
type embedded_array
  real r
  integer, pointer :: ia(:)
end type embedded_array

type(embedded_array) ea (3)
```

After you select the **View > Dive In All** command, TotalView displays all three **r** elements of the **ea** array as if it were a single array.

The **View > Dive in All** command can also display the elements of C array of structures as arrays.

As TotalView's array manipulation commands work on what's displayed and not what is stored in memory, you can operate on an array created by this command in the same manner as any other array. For example, you can visualize the array, obtain statistics about it, filter elements within it, and so on.

View > Expand All

Expands all trees that are not displaying all of their information. That is, this is equivalent to clicking every + icon within the Variable Window.

View > Collapse All

Collapses all trees. That is, this is the equivalent to clicking every – icon within the Variable Window.

View > Add to Expression List

Adds the variable or element to the list of variables in the **Expression List** Window. If that window isn't being displayed, this command opens it.

If you have selected text within the Variable Window, TotalView sends the selected text to the Expression List window.

For more information, see **Expression List Window** on page 219.

View > Undive

Moves up one level in the Variable Window's *dive list* so that you return to the data display previously shown in this window. This list is a history of the data you have displayed. This command is analogous to the "Back" button

in a browser in that it returns you to a previous position. Each time you “undive”, you move up one item in this list.

As an alternative, you can select the < icon on the right side above the data display.

For additional information, see **View > Redive** on page 131.

View > Undive All

Moves up to the top of the Variable Window’s *dive list* so that you return to the data display that existed when you first opened the Variable Window. The dive list is a history of the data that TotalView has displayed in this window.

As an alternative, you can select the |< icon within the Variable Window’s toolbar.

View > Redive

Moves down one level in the Variable Window’s *dive list* so that you return to places you “undove” from. The dive list is a history of the data TotalView has displayed in this window. This command is analogous to the “Forward” button in a browser in that it returns you to a position you previously returned from. Each time you “redive”, you move one level towards the bottom of the window’s dive list.

As an alternative, you can select the > icon on the right above of the data display.

For additional information, see **View > Undive** on page 130.

View > Redive All

Restores the Variable Window’s *dive list* so that you return to *bottom-most* places you “undove” from. The dive list is a history of the data TotalView has displayed in this window. That is, this command is equivalent to repeatedly selecting the Redive command until there are no longer any windows left to restore.

As an alternative, you can select the >| icon within the Variable Window’s toolbar.

View > Laminate > None

Changes the variable display so that the data is no longer laminated. This means that TotalView just shows you values residing in the current process; that is, it no longer shows you the variable’s values in all of the program’s processes. (*Laminated* means that TotalView is showing you the value of a variable in more than one process or thread.)

View > Laminate > Process

Displays the value of a variable in the first thread in each process in the share group. When debugging a parallel program, seeing the value of a variable in all processes at the same time can be extremely useful.

View > Laminate > Thread

Displays the value of a variable in all threads in the process. When debugging a threaded program, seeing the value of a variable in all threads at the same time can be extremely useful. You can only turn on thread lamination if the process has more than one non-manager thread, and the variable being displayed depends on the stack or registers. (A static or global vari-

able will have the same value in all threads, so there is no point seeing it laminated across the threads).

When matching stack frames to construct laminated displays, TotalView considers calls from different sites in the same function to be different.

View > Compilation Scope > Fixed

When selected, the scope in which TotalView looks for a variable is *fixed* as being the scope that existed when it displayed the Variable Window.

The Variable Window that TotalView creates contains information that allows TotalView to determine the scope in which a variable is valid. That is, the value that TotalView displays is locked to this scope. If a variable with the same name exists in another scope, TotalView will not display this second variable in the displayed Variable Window because its scope differs. For example, many programs reuse the variable *i* as the loop counter. A second example are the stack variables contained within recursive routines. In this case, each instance of the variable is locked to a different instance of the stack frame.

This behavior, which is the default, occurs when this command's button is selected. If you would like the scope to vary, select the **View > Compilation Scope > Floating** command.

View > Compilation Scope > Floating

When selected, TotalView looks in the current scope to determine if it contains a variable with the same name as the variable that was used when the Variable Window was first created. If TotalView finds one, it displays information about it in the Variable Window even though its scope is not the same as what originally existed.

For example, if you use the variable *i* as a loop counter in many places, the Variable Window will always display the in-scope variable when this is set. Similarly, if you are displaying Variable Windows for stack variables in a recursive function, the Variable Window will show information for the current recursive routine.

If you want the scope to be fixed, select the **View > Compilation Scope > Fixed** command.

View > Loader Symbols

When you click on a library or program name within the Program Browser Window, TotalView displays the data in a Variable Window. Normally, this information does not include loader symbols. Selecting this command tells TotalView to display these signals. Reselecting this command tells TotalView that it should no longer display this information.

For more information, see Chapter 6, "Program Browser Window," on page 155.

View > Padding

When selected, TotalView displays padding data that the compiler has added to data structures. (The compiler adds padding bits to data to force the alignment on to word boundaries.)

By default, TotalView does not display this information as errors caused by padding seldom occur. However, you may see errors due to padding if you miscast your data.

Tools Menu Commands

The following are on the **Tools** Menu:

- **Tools > Create Watchpoint** on page 133
- **Tools > Add to Expression List** on page 136
- **Tools > Block Properties** on page 136
- **Tools > Visualize** on page 138
- **Tools > Visualize Distribution** on page 138
- **Tools > Statistics** on page 138
- **Tools > Attach Subset (Array of Ranks)** on page 140

Tools > Create Watchpoint

Defines or modifies an unconditional data watchpoint, or a conditional data watchpoint.



TotalView only supports modify watchpoints. That is, TotalView only triggers the watchpoint if your program modifies the memory location. If the same value is written back to the location, the watchpoint does not trigger.

When a watchpoint triggers, the thread's PC points to the instruction *after* the instruction that caused the watchpoint to trigger. In some cases, where a memory store instruction is the last instruction in the source line, the PC will point to the source line *after* the source line that contains the triggering instruction

Platform Restrictions

The number of watchpoints, their size, and alignment restrictions differ from platform to platform. (This is because TotalView relies on the operating system and its hardware to implement data watchpoints.)



Watchpoints are not available on Alpha Linux and HP.

The following list describes constraints that are unique to each platform:

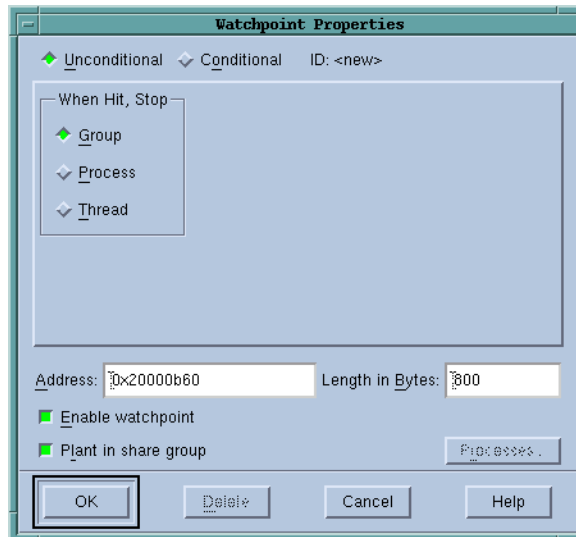
Computer	Constraints
HP Alpha Tru64	Tru64 places no limitations on the number of watchpoints that you can create, and no alignment or size constraints. However, watchpoints can't overlap, and you can't create a watchpoint on an already write-protected page. Watchpoints use a page-protection scheme. Because the page size is 8,192 bytes, watchpoints can degrade performance if your program frequently writes to pages that contains watchpoints
IBM AIX	You can create one watchpoint on AIX 4.3.3.0-2 (AIX 4.3R) or later systems running 64-bit chips. These are Power3 and Power4 systems. (AIX 4.3R is available as APAR IY06844.) A watchpoint cannot be longer than 8 bytes, and you must align it within an 8-byte boundary.
IRIX6 MIPS	Watchpoints are implemented on IRIX 6.2 and later operating systems. These systems let you create approximately 100 watchpoints. There are no alignment or size constraints. However, watchpoints can't overlap.

Computer	Constraints
Linux x86, Linux IA-64, Linux x-86-64 (AMD and Intel)	You can create up to four watchpoints and each must be 1, 2, or 4 bytes in length, and a memory address must be aligned for the byte length. That is, you must align a 4-byte watchpoint on a 4-byte address boundary, and you must align 2-byte watchpoint on a 2-byte boundary, and so on.
HU-UX IA-64	You can create up to four watchpoints. The length of the memory being watched must be a power of 2 and the address must be aligned to that power of 2; that is, (address % length) == 0 .
Solaris SPARC	TotalView supports watchpoints on Solaris 2.6 or later operating systems. These operating system let you create hundreds of watchpoints, and there are no alignment or size constraints. However, watchpoints can't overlap.

Typically, a debugging session does not use many watchpoints. In most cases, only one memory location at a time is being monitored. So, restrictions on the number of values you can watch are seldom an issue.

Watchpoint Commands

Figure 65: Tools > Watchpoint Dialog Box



Watchpoint type

Lets you indicate if the watchpoint is conditional or unconditional.

Unconditional

Tells TotalView to trigger the watchpoint whenever the memory location's value is modified. You can tell TotalView that it should also stop the thread's process or control group when the thread is stopped. Unconditional watchpoints are discussed later in this section.

Conditional

Tells TotalView that, after a memory location is modified, it should evaluate the condition. Conditional watchpoints are discussed later in this section.

Address

The first (or lowest) memory address to watch. Depending on the platform, this address may need to be aligned to a multiple of the **Byte Size** field. If you edit the address of an existing watchpoint, TotalView alters the watchpoint so it will watch this new memory location and reassigns the watchpoint's action point ID.

Byte Size

The number of bytes that TotalView should watch. Normally, this amount is the size of the variable. However, some architectures limit the amount of memory that can be watched. In other cases, you may want TotalView to monitor a few locations in an array. For information on architectural limitations, see "Platform Restrictions".

Enable watchpoint

If selected, TotalView makes this watchpoint active. (If a watchpoint is inactive, TotalView ignores changes to the watched memory locations.)

Plant in share group

If selected, TotalView creates a watchpoint for each thread in the share group. Some architectures place limits on the size and number of watchpoints. See "Platform Restrictions" for more information.

Unconditional Watchpoints

The only control unique to unconditional watchpoints is **When Hit, Stop**, which tells TotalView what should be stopped when a watched location changes. While TotalView will always stop the executing **Thread** (the thread of interest), it can also stop the thread of interest's control **Group** or all thread's in the thread of interest's **Process**.

Conditional Watchpoints

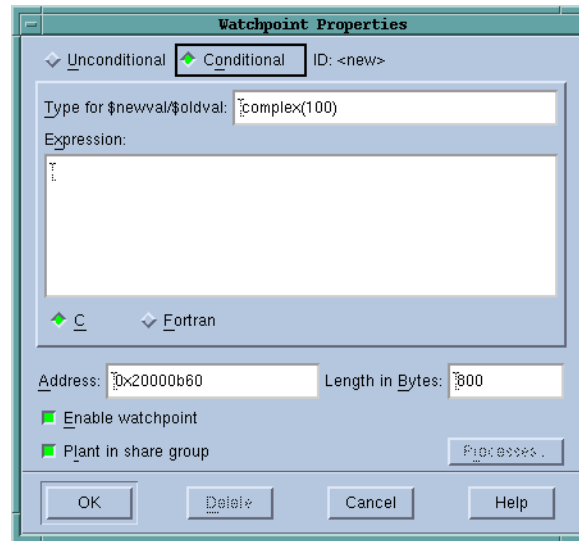
Using the following controls, you can create a watchpoint that triggers only when the condition you specify occurs.

Type for \$newval/\$oldval

If you will be placing the value stored at the memory location into a variable (using **\$newval** and **\$oldval**), you must define the variable's data by using a scalar type, such as **int**, **integer**, **float**, **real**, or **char**. You cannot use aggregate types such as arrays and structures.

If the size of the watched location matches the size of the data type entered here, TotalView interprets the **\$oldval** and **\$newval** information as the variable's type. If you are watching an entire array, the watched location can be larger than the size of this type.

Figure 66: Tools > Watchpoint Dialog Box



Expression Enter a code fragment. The expression is compiled into interpreted code that is executed each time the watchpoint triggers. These points can be used to implement countdown and conditional watchpoints. For additional information, see Setting Action Points of the *TotalView Users Guide*.

C or Fortran Indicates the programming language in which you wrote the expression.

The *TotalView Users Guide* contains additional information on watchpoints.

See “*Expression List Window Overview*” on page 219 for more information.

Tools > Add to Expression List

Tools > Block Properties

The **Memory Block Properties** Window displays information about all of the blocks that you asked the Memory Debugger to keep track of by using this command. You can only use this command if this variable’s memory block was allocated within the heap. For example, assume that you have the following statements within your program:

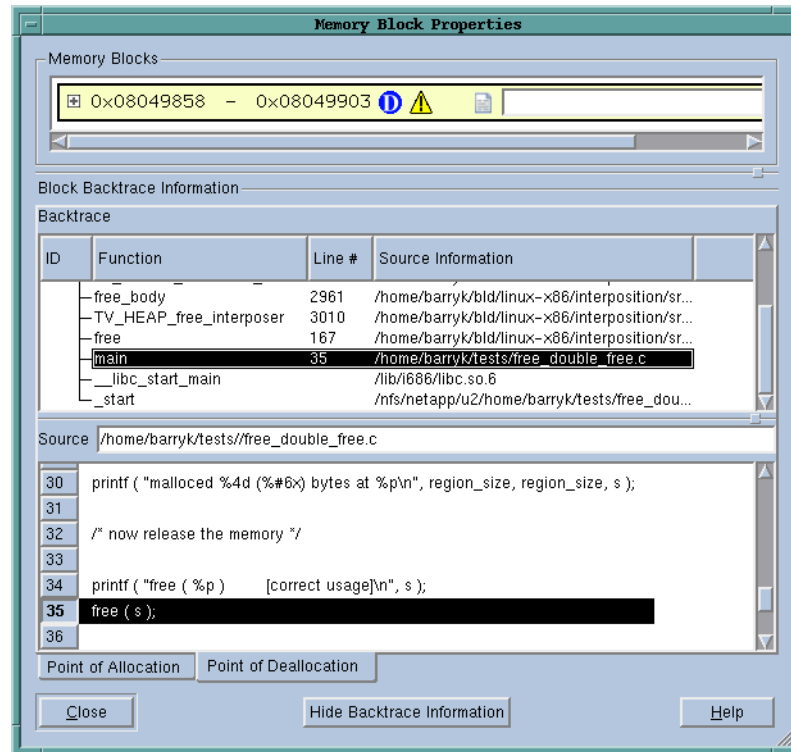
```
void *s;
s = malloc(sizeof(int)*200);
```

You cannot obtain block properties for variable `s` as `s` is allocated on the stack. However, after `malloc()` allocates memory, `s` points to a memory block. You can now have the Memory Debugger watch this allocated block.

This means that you will need to dive on a pointer so that TotalView can dereference the pointer before you use this command.

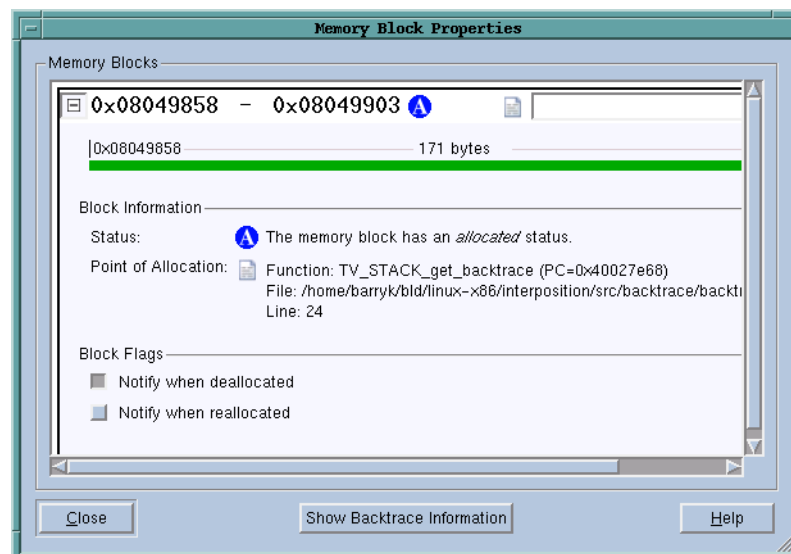
You can display this window in two different ways. Pressing the **Hide Backtrace Information** conceals most of what is displayed in Figure 67 on page 137.

Figure 67: Tools > Memory Block Properties Window



After pressing this button, the window reconfigures itself to that shown in Figure 68 on page 137. When this window is being displayed, pressing **Show Backtrace Information** shifts it back.



Figure 68: Tools > Memory Block Properties Window



As an alternative, you can both make the window larger and use the splitter control to enlarge the top area.

The information within the bottom area is essentially the same as that which is displayed when you generate a Backtrace View in the Memory Debugger. You will find information on the contents of the Block Backtrace Information area within “*Heap Status Page*” on page 191

The first line contains a graphical summary of the blocks information. It contains:

- Memory block extent: the numbers in bold indicate the starting and ending address of the block. For example, the block in Figure 68 indicates the block starts at 0x08049858 and ends at 0x08049903.
- Status: an indicator indicating if the block is allocated, deallocated, leaked, or being hoarded. For example, the  in Figure 68 indicates that the block is allocated.
- Point of Allocation: if you place your mouse over the  icon, TotalView displays a tool tip that shows the function in which the block was allocated, the function’s source file, and the line number within that file.
- Comment: if you have more than one or two memory blocks, you might want to enter a comment in the text box to remind yourself what the block is.

The next lines within the Memory Blocks area restate the information that was presented in the summary area. The color of the block’s graphic is the same as that used in the Heap Status Page’s Graphical View.

The Block Flags area contains two check boxes:

- Notify when deallocated
- Notify when reallocated

Checking a box tells the Memory Debugger that it should stop execution when the block is deallocated or reallocated and display the Memory Event Details Window. This allows you to track when your program is managing memory and what is being managed. For example, if you have a double free problem, obtaining notification lets you know where the block is first freed.

Tools > Visualize

Displays the TotalView Visualizer. The Visualizer lets you graphically display the values contained within a numeric array. For more information, see **The Visualizer Window** on page 145.

Tools > Visualize Distribution

Displays the TotalView Visualizer. The Visualizer lets you graphically display the values contained within a numeric array. This differs from a normal visualizer display in that one of the axis is the node upon which the data resides.

For more information, see **The Visualizer Window** on page 145.

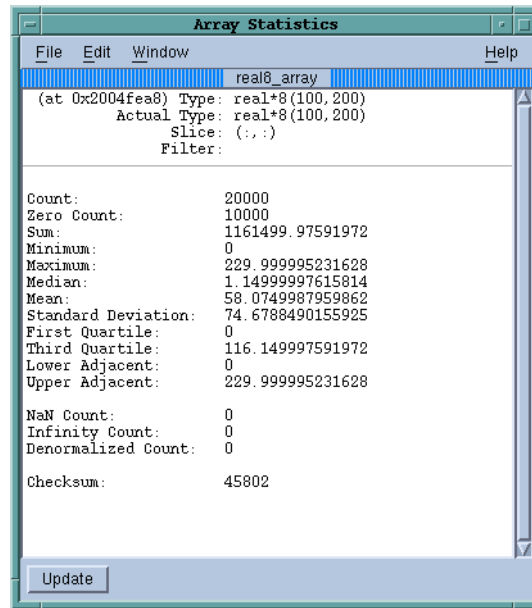
Tools > Statistics

Displays statistics for the displayed array elements. If the Variable Window contains a slice expression or a filter, only these values are used when TotalView creates the statistical information. (See Figure 69.)

The statistics calculated are:

- | | |
|------------------|---|
| Adjacents | Displays the lower and upper adjacents. The lower adjacent provides an estimate of the lower limit of the ar- |
|------------------|---|

Figure 69: Array Statistics



ray. Values below this limit are called *outliers*. The lower adjacent value is the first quartile value less 1.5 times the difference between the first and third quartiles.

The upper adjacent value provides an estimate of the upper limit of the array. Values above this limit are called outliers. The upper adjacent value is the third quartile value plus 1.5 times the difference between the first and third quartiles.

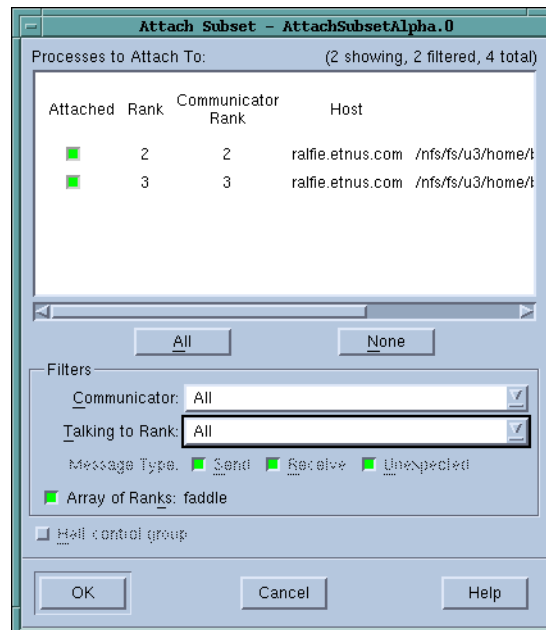
Checksum	A checksum value for the array elements.
Count	The number of elements that participated in the statistical calculations. For floating-point arrays, this does not include any NaN or infinity (INF) values.
Denormalized Count	A count of the number of denormalized values found in a floating-point array. This includes both negative and positive denormalized values as defined in the IEEE floating-point standard. These elements do participate in the statistical calculations.
Infinity Count	A count of the number of infinity (INF) values found in a floating-point array. This includes both negative and positive infinity as defined in the IEEE floating-point standard. These elements do not participate in statistical calculations.
Maximum	The largest value in the array.
Mean	The mean value of the array.
Median	The median value of the array.
Minimum	The smallest value in the array.

NaN Count	A count of the number of NaN values found in a floating-point array. This includes both signaling and quiet NaNs as defined in the IEEE floating-point standard. These elements do not participate in statistical calculations.
Quartile	Displays either the first or third quartile. The first quartile is the 25th percentile value in the array. That is, 25 percent of the array values are smaller than this value and 75 percent are greater. The third quartile is the 75th percentile value in the array. This means that 75 percent of the array values are less than this value and 25 percent are greater.
Standard Deviation	The standard deviation of the array values.
Sum	The summation of all the array elements.
Zero Count	The number of elements that have a value of 0.

Tools > Attach Subset (Array of Ranks)

Lets you indicate which processes TotalView should attach to when these processes begin executing. Limiting the processes to which TotalView attaches is beneficial as TotalView does not have to be concerned with unattached processes. That is, because you know that you will not be interested in a what goes on in within a process, you can cut down on the time that TotalView uses to attach to all or most of your processes.

Figure 70: Attach Subset Dialog Box



Processes to Attach To

Use the controls in this area to specify the processes to which TotalView should attach when they are created. You have three choices:

Selection Area

Individually select or deselect processes

All Attach to all of the listed processes.

None Do not attach to any of these processes.

After selecting **All** or **None**, you can individually select or unselect processes. That is, if you only want to select a couple of processes, begin by clicking **None**, then select the few to which TotalView should attach.

Filters You can restrict the list by selecting the controls in this area.

Communicator

The communicators within this list tell TotalView which processes it should display. Selecting one of the communicators contained within this list tells TotalView that it should only display processes using this communicator. You can then select or clear these values in one of the three ways just discussed.

Talking to Rank

TotalView will limit the graph to communicators that receive messages from the indicated ranks. In addition to your rank numbers, TotalView includes two special variables: **All** and **MPI_ANY_SOURCE**.

Message Type

TotalView will only show **Send**, **Receive**, or **Unrestricted** messages.

Array of Ranks

This checkbox is automatically selected by TotalView if you have invoked this command from the **Tools > Attach Subset (Array of Ranks)** command. If the Variable Window is displaying an array, invoking this command tells TotalView that the array's elements indicate ranks.

Halt control group

Selecting this button tells to stop all of the processes in the current process's control group after it attaches to a process. If it isn't selected, TotalView will immediately execute the control group after it attaches to them.

Window Menu Commands

The **Window** Menu contains the following commands:

- **Window > Update** on page 142
- **Window > Update All** on page 142
- **Window > Duplicate** on page 142
- **Window > Memorize** on page 142
- **Window > Memorize all** on page 142
- **Window > Root** on page 143

Window > Update

Updates the display of this window. *Update* means that the process is momentarily stopped so that TotalView can determine the program's state and the value of variables. It then updates this Variable Window and other Variable Windows associated with this variable's process so that they contain updated values.

Window > Update All

Tells TotalView to update the contents of all windows. That is, TotalView fetches and then displays the current value of the information in all open windows. *Update* means that the process is momentarily stopped so that TotalView can determine the program's state and the value of variables. It then updates this Variable Window and other Variable Windows associated with this variable's process so that they contain updated values.

Window > Duplicate

Tells TotalView to create an identical copy of this Variable Window.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

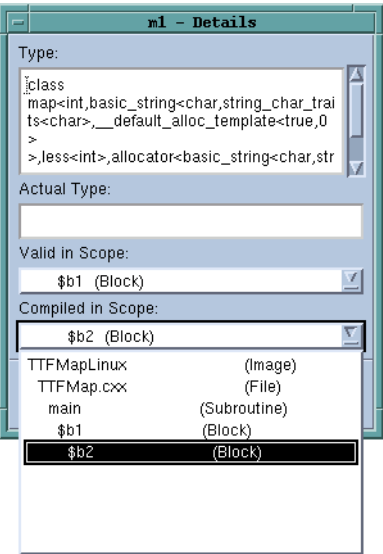
If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root Tells TotalView to bring the Root Window to the top of the display.

Variable Details Window

If you select the Details button, TotalView displays a window containing additional information about the variable.

Figure 71: Details Window



Here what you will see:

- | | |
|--------------------------|--|
| Type | Contains the variable's data type. If the name is too long to fit into the Variable Window's Type field, you'll be able to see its entire name here. |
| Actual Type | If there is a transformation of some kind (not a cast, however) so that the Type is actually something different, tells you what it's underlying implementation is. For example, if TotalView is displaying an STL vector as an array, it's type will be "array", while its actual type is "vector". |
| Valid in Scope | Indicates the scope in which the variable resides. That is, the variable is "valid" anytime the PC is within this scope. If you display the pulldown list, you will see the complete scope specification. |
| Compiled in Scope | Indicates the scope in which the expression was compiled. If you alter the text within the Expression field of the Variable Window, this field lets you know the scope where the expression is defined. If you display the pull- |

down list, you will see the complete scope specification.

The Visualizer Window

4

The **Tools > Visualize** and **Tools > Visualize Distribution** commands, and the **\$visualize** intrinsic send an array's values to the TotalView Visualizer so that it can display these values. Laminated data panes can also be visualized, in which case, the process or thread index forms one axis of the display.

The **Launch Strings Page** within the **File > Preferences Window** dialog box allows you to specify the command TotalView will use to start the Visualizer.

The **Maximum array rank** field sets the maximum rank of the array that TotalView will export to the Visualizer. The Visualizer cannot visualize arrays of rank greater than two; if you are using another visualizer, however, or if you are dumping binary data, you can set the limit here.

The Visualizer can be used in two ways: it can be launched by TotalView to visualize data as you debug your programs, and it can be run from the command line to visualize data dumped to a file in a previous TotalView session.

Visualizing your program's data uses two interactions:

- You interact with TotalView to choose *what* you want to visualize and *when* it should make snapshots of your data.
- You interact with the visualizer to choose *how* you would like your data to be displayed.

The TotalView debugger handles the first of these interactions, extracting data and marshalling it into a standard format that it sends down a pipe. The Visualizer then reads the data from this pipe and displays it for analysis.

The Visualizer has two types of windows:

- *Directory Window*

A single main window lists the datasets that you can visualize. You can use this window to set global options and to create views of your datasets.

- *Data Window*

A *data window* contain images of the datasets. By interacting with a Data Window, you can change its appearance and set dataset viewing options.

Using the Directory Window, you can open several Data Windows on a single dataset to get different views of the same data.

For more information on the Visualizer, see *TotalView Users Guide*.

Directory Window

The Directory Window contains a list of the datasets you can display.

You can select a dataset by left-clicking on it, and you can only select one dataset at a time. The **View** menu commands let you select **Graph** or **Surface** visualizations. Whenever TotalView sends a new dataset, the Visualizer updates its list of datasets. To delete a dataset from the list, use the **File > Delete** command.

File Menu Commands

File > Delete

The following commands are on the **File** pulldown:

- **File > Delete**

- **File > Exit**

Deletes the currently selected dataset. It removes the dataset from the dataset list and *destroys* any Data Windows displaying it.

File > Exit

Closes all windows and exits the **Visualizer**.

View Menu Commands

View > Graph

The following commands are on the **View** pulldown:

- **View > Graph**

- **View > Surface**

Creates a new graph window.

View > Surface

Creates a new surface window.

Options Menu Command

Options > Auto Visualize

The following command is on the **Options** pulldown: *Options > Auto Visualize*.

This item is a toggle; when enabled, the **Visualizer** automatically visualizes new datasets as they are read.

Data Window

Data Windows display graphical images of your data. Every Data Window contains a menu bar and a drawing area. The Data Window title is its dataset identification.

The **File** menu on the menu bar is the same for all Data Windows. Other items on the menu bar are specific to particular types of Data Window.

File Menu Commands

The following commands are on the **File** pulldown:

- **File > Directory**
- **File > New Base Window**
- **File > Options**
- **File > Delete**
- **File > Close**

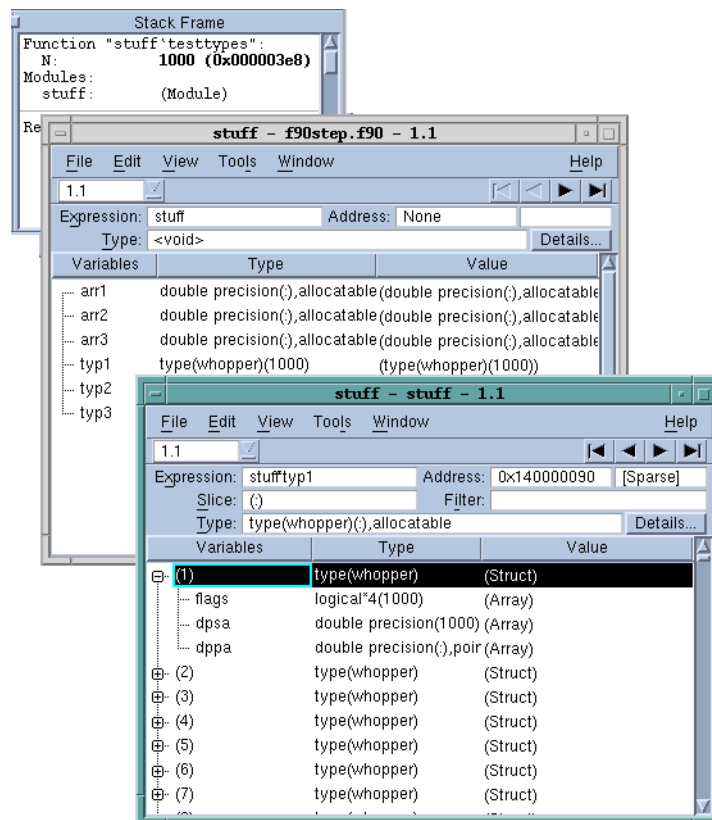
File > Directory	Raises the Directory Window to the front of the desktop. If the Directory Window is minimized, it is restored.
File > New Base Window	Creates a new Data Window using the same visualization method and dataset as the current Data Window.
File > Options	Pops-up window of viewing options. This window has a <i>control</i> area and an <i>action</i> area. The control area is specific to the Data Window. The action area contains standard OK , Apply , and Cancel buttons.
File > Delete	Deletes the Data Window's dataset from the dataset list. This also destroys any other Data Windows viewing the dataset.
File > Close	Closes the Data Window.

Fortran Modules Window

5

Selecting the Process Window's **Tools > Fortran Modules** command tells TotalView to display a window containing information about the Fortran modules that are used by a process.

Figure 72: Fortran Modules





If you compiled your code using the SUNPro F90 compiler, TotalView must scan all debugging information to locate all module names. As this can be very time-consuming, TotalView only displays information for modules that are named within already read symbol information. If module you want to see is not displayed, use the Process Window's **View > Lookup Function** command to open a file using a module.

To see more information about a module, dive (double-click) on the module's name. TotalView responds by displaying a Variable Window containing information about that module's variables.

If a module already has a window when you dive into its name in the Fortran Modules Window, TotalView brings its old window to the front of the screen.

To see information about a module without first scrolling to it in the modules Window, select the Process Window's **View > Lookup Variable** command and enter the name of the module into the displayed dialog box.

File Menu Commands

The commands on the **File** pulldown are:

- **File > Close Similar** on page 150
- **File > Close** on page 150

File > Close Similar

Closes this windows and close other Fortran Module Windows.

File > Close

Closes this window. No other windows are affected by this command.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 150
- **Edit > Cut** on page 150
- **Edit > Copy** on page 151
- **Edit > Paste** on page 151
- **Edit > Delete** on page 151
- **Edit > Find** on page 151
- **Edit > Find Again** on page 152

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Win-

dow's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

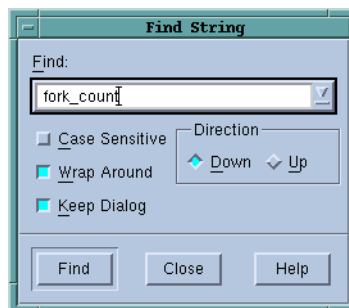
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 73: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |

Wrap On Search	If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option.
Keep Dialog	If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box.
<i>Direction</i>	Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the Find box.
Close	Closes the Find dialog box.

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** pulldown are:

- **View > Dive** on page 152
- **View > Dive in New Window** on page 152

View > Dive

Tells TotalView to "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Window Menu Commands

The commands on the **Window** pulldown are:

- **Window > Update** on page 153
- **Window > Update All** on page 153
- **Window > Memorize** on page 153
- **Window > Memorize all** on page 153
- **Window > Root** on page 153

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

Program Browser Window

6

Use this window to examine symbols contained within your program. After you invoke this command, TotalView displays a window containing the libraries and programs that make up your executable. If you dive on any of these, TotalView displays a Variable Window that contains information about the library or program. (See Figure 74 on page 156.)

If you need to see more information about a variable, you can dive on it. In this case, the variable you dove upon replaces the information in the current Variable Window.

If you need to see loader symbols (they are not displayed by default), use the **View > Loader Symbols** command from within the Variable Window.

File Menu Commands

The **File** menu contains the following commands:

- **File > Close Similar** on page 155
- **File > Close** on page 155
- **File > Exit** on page 155

File > Close Similar

Closes all Program Browser Windows.

File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Figure 74: Program Browser

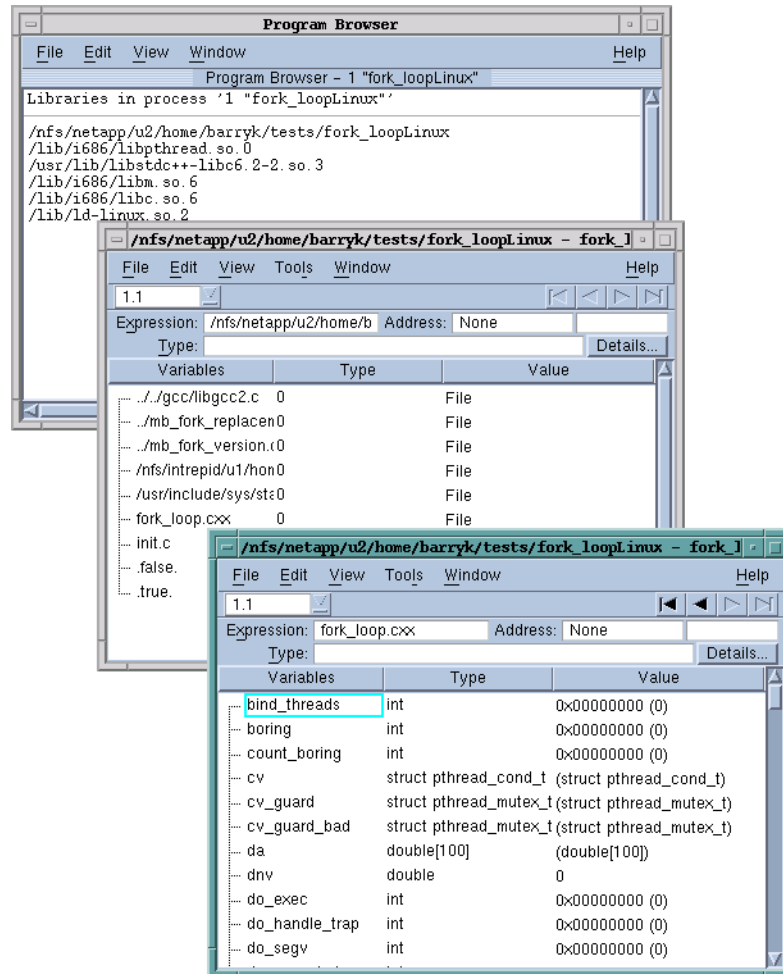
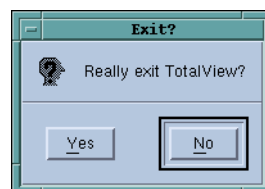


Figure 75: File > Exit Dialog Box



Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 157
- **Edit > Cut** on page 157
- **Edit > Copy** on page 157
- **Edit > Paste** on page 157
- **Edit > Delete** on page 157
- **Edit > Find** on page 158
- **Edit > Find Again** on page 158

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 76: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. By selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option. |
| Keep Dialog | If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box. |
| <i>Direction</i> | Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file." |
| Find | Tells TotalView to search for the text within the Find box. |
| Close | Closes the Find dialog box. |

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the View menu are:

- **View > Dive** on page 159
- **View > Dive in New Window** on page 159

View > Dive

Tells TotalView to open a “dive” into the selected item. In all cases, “dive” means that TotalView will show more information. TotalView responds by opening a Variable window containing information about that variable.

If the Variable Window already exists for the variable, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a “dive” into the selected item. In all cases, “dive” means that TotalView will show more information. TotalView responds by opening a Variable window containing information about that variable.

Even if a Variable Window already exists for the variable, TotalView will still create a new window for this information.

Window Menu Commands

The commands on the window pulldown are:

- **Window > Update** on page 159
- **Window > Update All** on page 159
- **Window > Memorize** on page 159
- **Window > Memorize all** on page 160
- **Window > Root** on page 160

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program’s state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program’s state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window’s position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the “Force Windows Position” option within the **File > Preference’s Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

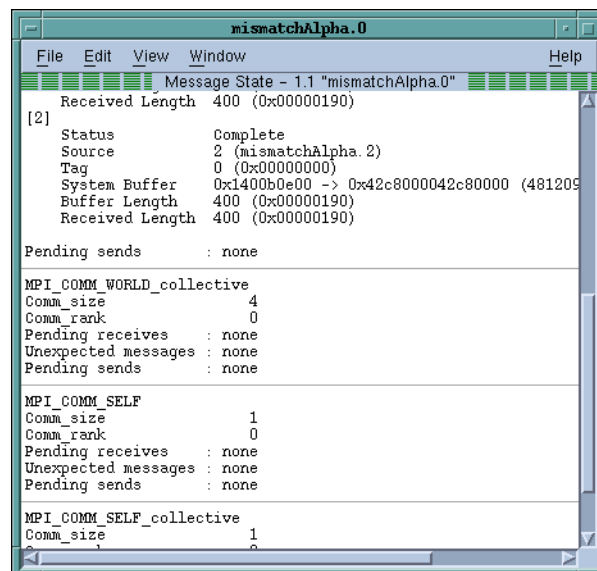
Message Queue Window

7

Message Queue Window Overview

The **Tools > Message Queue** command tells TotalView that it should display information about the current process's message queues.

Figure 77: Message Queue Window



Message queues are displayed for the following versions of MPI:

- MPICH version 1.1.0 or later.
- Compaq Alpha MPI (DMPI) version 1.7.
- HP HP-UX version 1.6.
- IBM MPI Parallel Environment (PE) version 2.3 or 2.4; but only for programs using the threaded IBM MPI libraries. TotalView cannot show message queues for earlier releases or with the non-thread-safe version of the IBM MPI library. Therefore, to use TotalView's message queue display

with IBM MPI applications, you must compile and link your code using the **mpcc_r**, **mpxlf_r**, or **mpxlf90_r** compilers.

- For SGI MPI TotalView message queue display, you must obtain the Message Passing Toolkit (MPT) release 1.3 or later. Check with SGI for availability.

This dialog box displays the state of each of the MPI communicators that exist in the process. In some MPI implementations, such as MPICH, user-visible communicators are implemented as two internal communicator structures, one for point-to-point and the other for collective operations. TotalView displays both.



You cannot edit any of the fields in the Message Queue dialog box.

The contents of the Message Queue dialog box are only valid when a process is stopped.

For each communicator, TotalView displays the following fields:

Communicator Name

MPI lets you name predefined communicators such as **MPI_COMM_WORLD()**. In addition, MPICH 1.1 and Compaq MPI use the MPI-2 **MPI_NAME_PUT()** and **MPI_NAME_GET()** communicator naming functions that let you associate a name with a communicator. If you use **MPI_NAME_PUT()** to name a communicator, TotalView uses the name you gave it when it displays the communicator.

IBM MPI and SGI MPI do not implement the MPI-2 communicator naming functions, which means that only predefined communicators are named. For user-created communicators, TotalView displays the integer value that represents the communicator. This is the value that a variable of type **MPI_Communicator** has when it represents a communicator.

Comm_size

The number of processes in the communicator. This value is the same value as occurs when you apply **MPI_Comm_size()** to the communicator.

Comm_rank

The rank in the communicator of the process that owns the Message Queue Window. This information is the same information you would get if you had applied **MPI_Comm_rank** to the communicator in this process.

Pending receive operations

A list of pending receive operations.

Unexpected messages

A list of messages sent to this communicator but that have not yet been matched with a receive.

Pending send operations

A list of pending send operations.

Message Operations

For each communicator, TotalView displays a list of pending receive operations, unexpected messages, and pending send operations. Each operation has an index value displayed in brackets (*[n]*), and each operation can include the following fields:

Actual Source	If the Status is Complete and the Source is ANY , this is the receiving process.
Actual Tag	If the Status is Complete and the Tag value is ANY , this is the received tag value.
Buffer Length or Received Length	The buffer length in bytes, shown in decimal and hexadecimal.
Function	The MPI function (IBM MPI only). The name of the MPI function associated with the operation; for example, MPI_Irecv() .
Source or Target	The source or target process. Source is the process from which the message should be received. Target is the process to which the message is being sent. This field shows the index of the process in the communicator, and the process name in parentheses. The display shows ANY if the message is being received from MPI_ANY_SOURCE . Dive into this field to display a Process Window.
Status	The status of the operation. Operation status can be Pending , Active , or Complete .
Tag	The tag value. If the message is being received with MPI_ANY_TAG , the display shows ANY .
Type	The MPI data type (IBM MPI only). The MPI data type associated with the operation; for example, MPI_INT() .
User Buffer, System Buffer, or Buffer	The address of the buffer. Dive into this field to view a Variable Window displaying the buffer contents.

File Menu Commands

The commands on the **File** Menu are:

- **File > Close Similar** on page 163
- **File > Close** on page 163
- **File > Exit** on page 164

File > Close Similar

Closes this window and closes windows whose contents are similar to this window.

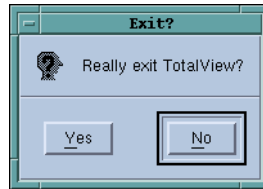
File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

Figure 78: File > Exit Dialog Box



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 164
- **Edit > Cut** on page 164
- **Edit > Copy** on page 164
- **Edit > Paste** on page 165
- **Edit > Delete** on page 165
- **Edit > Find** on page 165
- **Edit > Find Again** on page 166

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

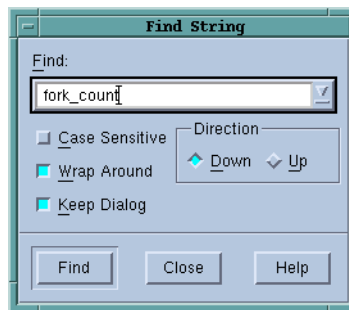
You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 79: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option. |
| Keep Dialog | If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box. |

<i>Direction</i>	Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the Find box.
Close	Closes the Find dialog box.

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** Menu are:

- **View > Dive** on page 166
- **View > Dive in New Window** on page 166

View > Dive

Tells TotalView to "dive" into the selected item. In all cases, "dive" means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a "dive" into the selected item. In all cases, "dive" means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Window Menu Commands

The commands on the **Window** pulldown are:

- **Window > Update** on page 167
- **Window > Update All** on page 167
- **Window > Memorize** on page 167
- **Window > Memorize all** on page 167
- **Window > Root** on page 167

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

PVM Tasks Window



Selecting the **Tools > PVM Tasks** command within the Root Window tells TotalView to display a window that contains current information about PVM tasks and hosts. TotalView automatically updates this information as it receives events from PVM.

Figure 80: PVM Window

PVM Tasks And Configuration					
HOST	TID	PTID	PID	FLAG	EXECUTABLE
rsmp	40008	0	31660	404	-
rsmp	40009	0	25790	4	-
rsmp	4000a	40009	37828	4	mtile
albacore	80002	40009	641	6	mtile
albacore	80001	40003	2602	6	mtile

HOST	DTID	ARCH	SPEED
albacore	80000	SUN4SOL2	1000
rsmp	40000	AIX46K	1000

This window contains two areas. The top area lists the tasks and the bottom area lists the hosts. The top task area is further subdivided to define control groups. The information in the top task area is:

HOST	The name of the host upon which a task is executing.
TID	The parent process's task ID.
PTID	The UNIX process ID.
FLAG	The PVM message tag.
EXECUTABLE	The name of the executable file.

The information in the bottom host area is:

HOST	The name of a host upon which a task is executing.
DTID	The daemon's task ID.

ARCH	The architecture of the computer upon which the task is executing.
SPEED	Your speed setting.

You can attach to a PVM or DPVM task if:

- The machine architecture on which the task is running is the same as the machine architecture on which TotalView is running.
- The task must be created. (This is indicated when flag 4 is set in the PVM Tasks Window.)
- The task must not be a PVM tasker. If flag 400 is clear in the PVM Tasks Window, the process is a tasker.
- The executable name must be known. If the executable name is listed as a dash (-), TotalView cannot determine the name of the executable. (This can occur if a task was not created using the `pvm_spawn()` call.)

If the task to which you attached has related tasks that can also be debugged, TotalView asks if you want to attach to these related tasks. If you answer **Yes**, TotalView attaches to them. If you answer **No**, it only attaches to the task you dove on.

After attaching to a task, TotalView looks for attached tasks that are related to this task; if there are related tasks, TotalView places them in the same control group. If TotalView is already attached to a task you dove on, TotalView simply opens and raises the task's Process Window.

File Menu Commands

The commands on the **File** pulldown are:

- **File > Close** on page 170
- **File > Exit** on page 170

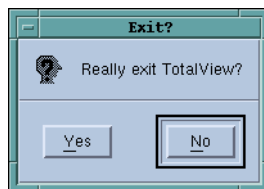
File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

Figure 81: File > Exit Dialog Box



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 171
- **Edit > Cut** on page 171
- **Edit > Copy** on page 171
- **Edit > Paste** on page 171
- **Edit > Delete** on page 171
- **Edit > Find** on page 172
- **Edit > Find Again** on page 172

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 82: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. By selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option. |
| Keep Dialog | If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box. |
| <i>Direction</i> | Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file." |
| Find | Tells TotalView to search for the text within the Find box. |
| Close | Closes the Find dialog box. |

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** pulldown are:

- **View > Dive** on page 173
- **View > Dive in New Window** on page 173

View > Dive

Tells TotalView to “dive” into the selected item. In all cases, “dive” means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

View > Dive in New Window

Tells TotalView to open a “dive” into the selected item. In all cases, “dive” means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Window Menu Commands

The commands on the **Window** pulldown are:

- **Window > Update** on page 173
- **Window > Update All** on page 173
- **Window > Memorize** on page 173
- **Window > Memorize all** on page 174
- **Window > Root** on page 174

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program’s state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program’s state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window’s position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

Memory Debugging Window

9

Memory Debugging Window Overview

When you configure the Memory Debugger or display a view, the action that the Memory Debugger takes is based on the processes that you select on the left side of the window. (The figure on the next page shows this window.)

The controls in the **Generate View** area tell the Memory Debugger which view to create on the right side of the window. This information is called a *view* because the Memory Debugger just shows a part of the information contained in the Memory Debugger tracking agent.

Process Set Selection

Configuring the Memory Debugger tells it which processes to track and what actions to perform. For example, the Memory Debugger Window shown on the next page can track more than one program. One of these programs has more than one process. If you select three processes out of the nine processes in this window, a leak detection view only shows leaks from these three processes. It ignores leaks in other processes.



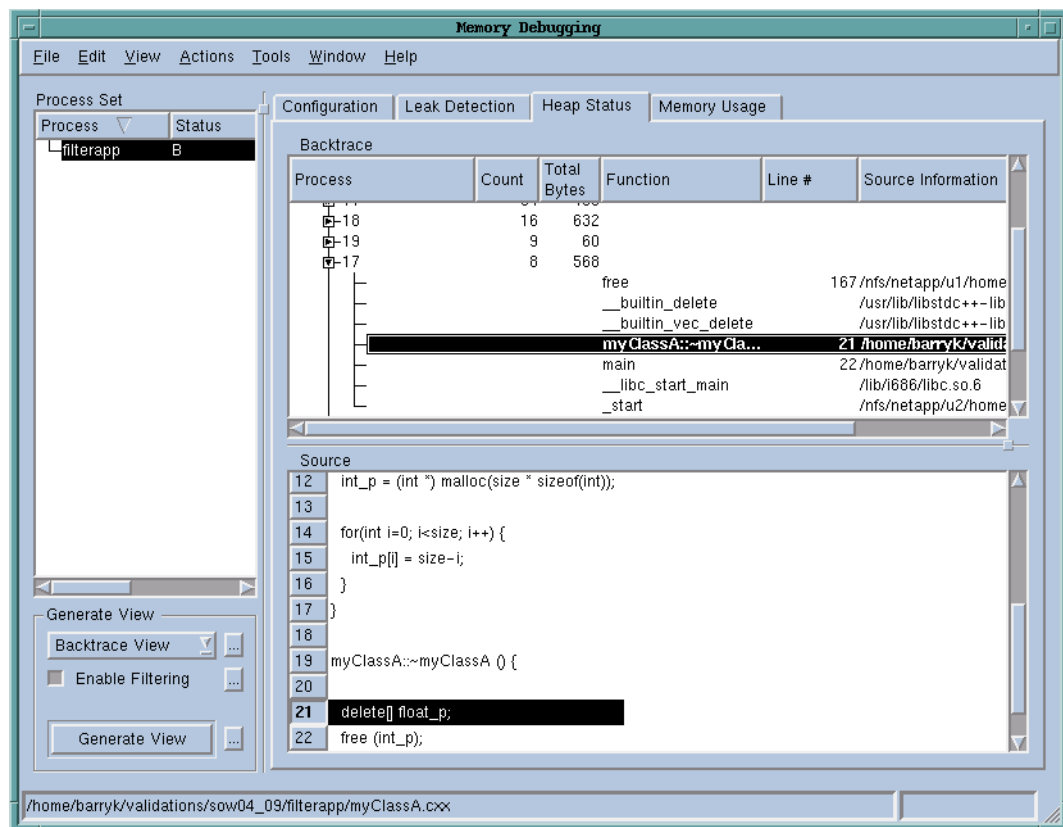
Be careful how many processes you select. With large multiprocess programs, you might be asking the Memory Debugger to process and analyze an enormous amount of data. In most cases, if you select one or two significant processes, you'll receive the information you need. Although the process of generating a view is lengthy, you can redisplay the information quickly after the Memory Debugger creates it.

Generate View Area

When you are viewing any page except the Configuration Page, you must tell the Memory Debugger which view it should display. (Specifying a view tells the Memory Debugger how it should display its information.) The controls in this area of the window are as follows:

Pulldown list Select a view from this list. Clicking on the arrow on the right side of this list displays your choices. This pull-

Figure 83: The Memory Debugger



down is not active when the Memory Debugger is displaying the Configuration Page.



Click this button to display a dialog box that contains preferences that modify or affect a view. The discussion of that page in other sections of this chapter describes these preferences.

Enable Filtering

Selecting this check box tells the Memory Debugger to apply filters to the information it is displaying. For additional information, see “**Tools > Filters**” on page 200.



Click this button to display the **Data Filters** Dialog Box. For more information, see “**Tools > Filters**” on page 200.

Generate View

After you select a view, pressing this button tells the Memory Debugger to display it.



If you need to save the information contained within a view, select this button. The Memory Debugger responds by displaying a dialog box that lets you write this information to disk. For more information, see “**Saving Views**” on page 177.

Rows and Columns

If a page displays information in columns, you can resize columns, change the column order, and control which columns the Memory Debugger displays, as follows:

- To resize a column, place the mouse pointer over the vertical column separator in the header. Press your left mouse button and drag the separator so that you've made the column as wide or as narrow as you want it to be. After you finished dragging the separator, release the left mouse button. The following figure shows the second column being made wider:

Figure 84: Resizing

Bytes	Count	End Address	Begin Address	Backtrace ID	Flags
-------	-------	-------------	---------------	--------------	-------

If you double-click on a separator, the Memory Debugger readjusts all widths.

- To change the column order, place your mouse pointer in a column header, press your left mouse button, and then drag the column to its new position. After it is in its new position, release the left mouse button. In the following example, the **Begin Address** column is being moved to the left:

Figure 85: Changing Position

Process	Bytes	Count	Begin Address	End Address	Backtrace ID	Flags
---------	-------	-------	---------------	-------------	--------------	-------


- To tell the Memory Debugger to hide a column or display a column you previously hid, right-click anywhere in the column header area. From the displayed context menu, click on an entry. If the entry is hidden, the Memory Debugger displays it. If the column is displayed, the Memory Debugger hides it. The following figure shows this context menu:

Figure 86: Displaying and Hiding Columns

Process	Bytes	Count	End Address	Begin Address	Backtrace ID	Flags
global_leak	450	9				
global...	450	9				
main	450	9				
L	450	9				
	90	1	...4519122	...45191		one
	80	1	...4519024	...4518:		one
	70	1	...4518934	...4518:		one
	60	1	...4518860	...4518:		one

- To tell the Memory Debugger to sort a column, click on the column heading. You can only sort some columns.

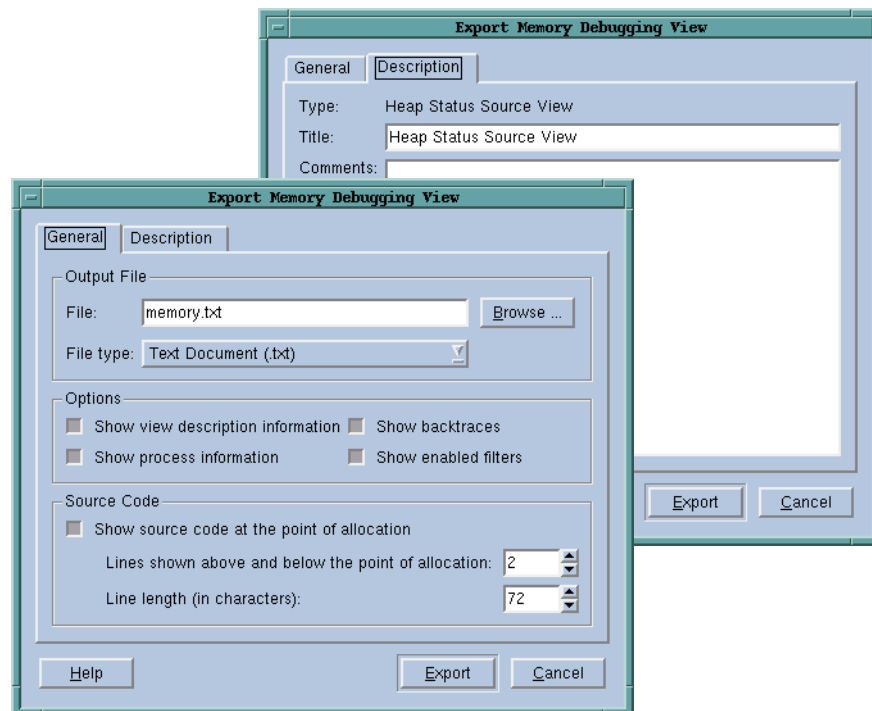
Saving Views

If you need to save the information within a view, press the  button, which is immediately to the left of the **Generate View** button. The Memory

Saving Views

Debugger responds by displaying a dialog box with two tabs. Both tabs are shown in Figure 87 on page 178:

Figure 87: Saving Views



General Page

The General Page contains the controls that let you specify what you want written. Here is what these controls do:

- Output File** The controls within this area tell the Memory Debugger where it should write memory information.
 - File** Enter the name of the file being created. You can change this from its default value by editing the text.
 - Browse** Press this button to display a dialog box that lets you select the directory in which the Memory Debugger will write the file.
 - File type** Select a file type. At version 6.7, the only format you can select is text.
- Options** The controls within this area tell the Memory Debugger what additional information it should write into the file.
 - Show view description information** When selected, the Memory Debugger writes information about the view type, data displayed, the user creating the file, and the host, date, and the comment recorded in the Description Page.
 - Show process information** When selected, the Memory Debugger writes information about the processes that were selected when you generate the view.

Show backtraces

When selected, the Memory Debugger writes stack backtrace information for the memory allocations in the view. If the view being displayed already contains backtraces, the Memory Debugger ignores this option.

Selecting this option increases the time the Memory Debugger needs to create the report. In addition, the size of the created file will be much larger.

Show enabled filters

When selected, the Memory Debugger names and describes the filters used when it generated the view

Source Code The Memory Debugger can also display lines from your source code.

Show source code at the point of allocation

When selected, the Memory Debugger displays source code information.

Lines shown above and below the point of allocation

Tells the Memory Debugger how many lines of source code above and below the allocation statement should also be displayed.

Line Length (in characters)

Tells the Memory Debugger how many characters it should use in each line when displaying information. Lines that are longer than this length are truncated.

Description Page

If you are writing a number of files, adding comments can help you identify the report. You can enter the following information:

Title	If the default title isn't what you want, enter something more descriptive here.
Comments	Enter text that describes the view information being written to disk.

Other topics that contain information are:

- **Configuration Page** on page 179
- **Leak Detection Page** on page 187
- **Heap Status Page** on page 191
- **Memory Usage Page** on page 195

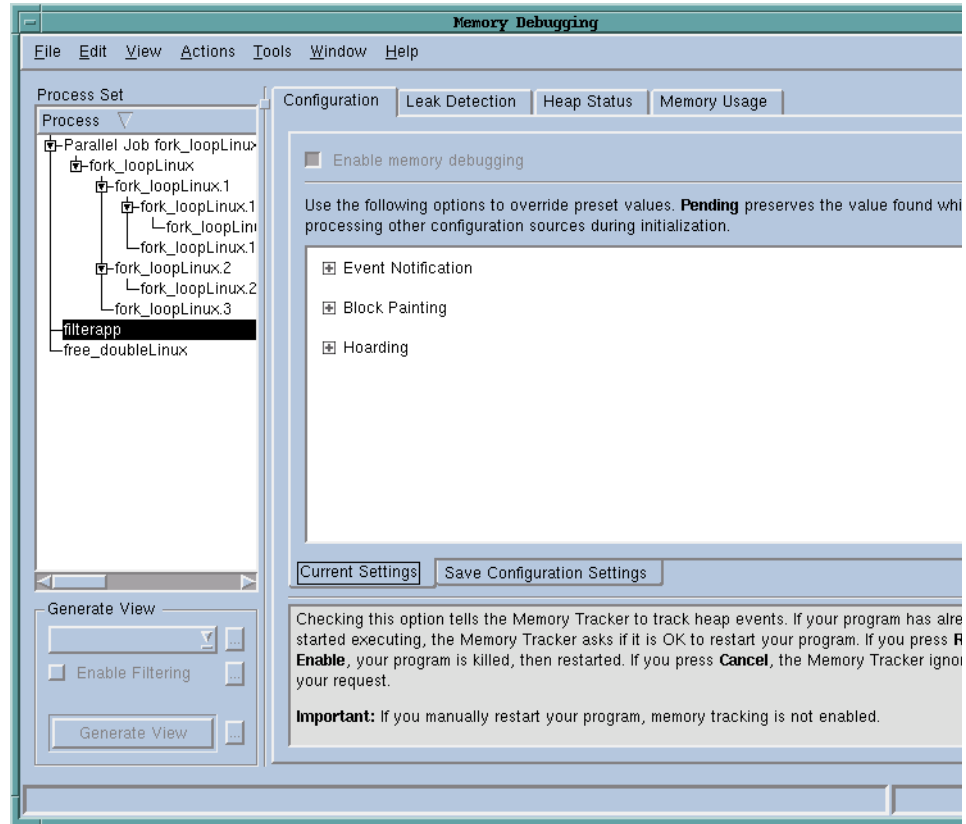
Configuration Page

The controls on the Configuration Page direct the actions that the Memory Debugger performs. They also allow you to save and restore settings that you have saved to disk. Topics in this section are:

- **"Current Settings Page"** on page 180
- **"Save Configuration Page"** on page 185

The following figure shows this page:

Figure 88: Configuration Page



Current Settings Page

Page The current settings page is where you tell the Memory Debugger which actions it should take when memory events occur. In addition, you can tailor these actions to your needs.

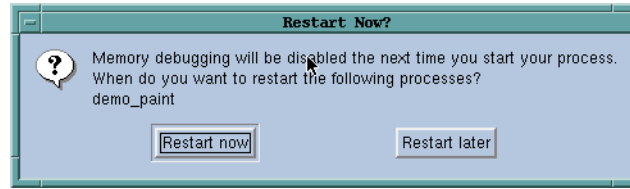


While you must explicitly tell the Memory Debugger to track your program's use of the heap API, you do not need to enable memory debugging to obtain a Memory Usage View.

The **Enable memory debugging** check box tells the Memory Debugger if it should track your program's use of the heap API. If TotalView can dynamically enable memory debugging, selecting this button loads the Memory Debugger. Most computing architectures do allow TotalView to enable the Memory Debugger before your program begins executing. However, TotalView cannot directly enable programs that run on an IBM RS/6000 or which run remotely. See “**Other Topics**” on page 229 for more information.

You cannot enable or disable the Memory Debugger while your program is executing. If you try, the Memory Debugger opens a dialog box asking if it should restart your program.

Figure 89: Restart Now Dialog Box

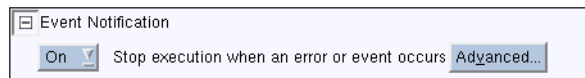


The third line of this error message has the name of the program or process that must be restarted.

Event Notification

If a memory event occurs using a function within the heap API, the Memory Debugger can tell TotalView to stop the program's execution so that you can determine the source of the event.

Figure 90: Memory Error Notification Area



Here is a description of the controls in this section:

Stop execution when an event or error occurs

Checking this box tells the Memory Debugger to stop program execution and display a dialog box when it detects that an event occurred that is related to using the heap API.

You can turn notification on and off both before and while your program is executing.

Advanced

Selecting this button tells the Memory Debugger to display a dialog box from which the events for which the Memory Debugger will stop execution. (See Figure 91 on page 182.)

By default, notification occurs for all events. You can individually turn an event off if you need to.

When an event occurs, the Memory Debugger stops program execution and tells TotalView to display its **Memory Event Details** window. (See Figure 92 on page 182.)

For information on this window, see "**Tools > Memory Event Details**" on page 109.

Figure 91: Event Types Dialog Box

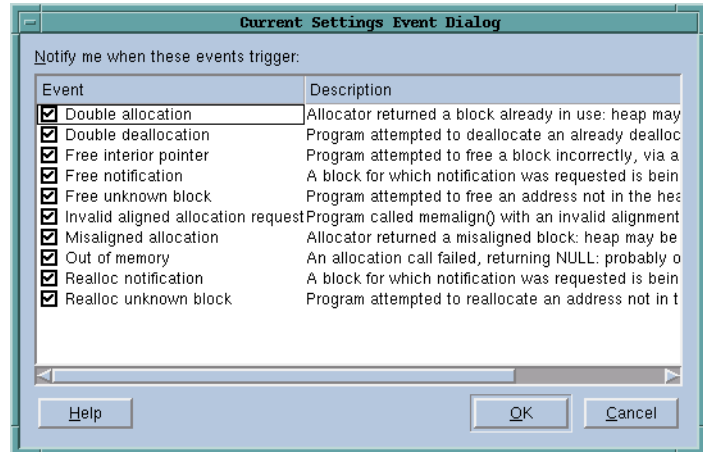
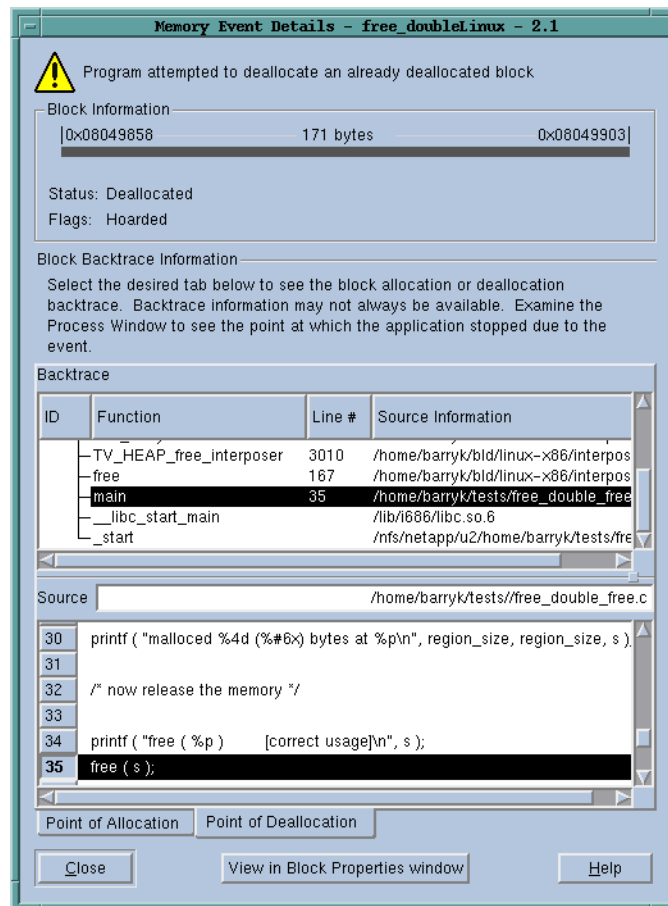


Figure 92: Memory Error Details Window

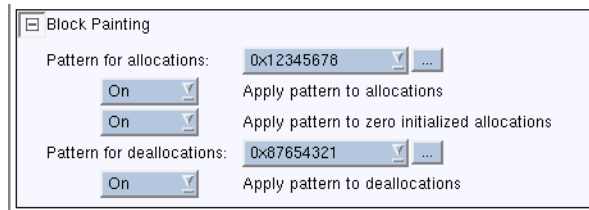


Block Painting

When you enable memory block painting, the Memory Debugger writes a bit pattern into newly allocated and newly deallocated heap memory

blocks. For information on using block painting, see “**Block Painting**” on page 31.

Figure 93: Memory Block Painting Area



Here is a description of these controls:

Pattern for allocations

The Memory Debugger uses the bit pattern in this box when it paints heap memory that was just deallocated. It uses the same pattern for normal allocations and zero-initialized allocations, which are allocations created by functions such as **calloc()**. The pulldown list contains patterns that you used previously.


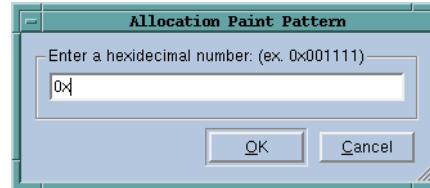
When you click the  button to the right of the pattern pulldown list, the Memory Debugger displays a dialog box into which you can type a new pattern:

Figure 94: Allocation Paint Pattern Dialog Box



If your program has not started executing, the Memory Debugger might not be able to display a pattern. If it cannot display a pattern, it displays **<pending>**.

You can change this pattern at any time and as many times as you want while your program is executing. Changing the pattern can help you identify when your program allocated a memory block. For example, when you see a pattern, you can tell if it was painted before or after you made a change.

If a data value uses more bits than indicated by the paint pattern, TotalView interprets the value using the number of bytes that the variable uses, not the number of bytes in the paint pattern. This means that you might need to cast the displayed value.

If you uncheck this box, the Memory Debugger stops painting allocated memory. You can recheck this box at a later time without having to restart your program.

Apply pattern to allocations

When **On** is selected, the Memory Debugger paints allocated memory using the bit pattern shown in the **Pattern for allocations** text field.

Apply pattern to zero initialized allocations

When **On** is selected, the Memory Debugger paints allocated memory that is set to zero by calls such as **calloc()** using the bit pattern shown in the **Pattern for allocations** text field.

You cannot paint zero-allocated memory unless you are also painting normal allocations. If you set the **Apply pattern to allocations** to **Off**, the Memory Debugger also sets this control to **Off**.



Setting this option to On can break your program if you depend upon the allocated memory being set to zero.

Pattern for deallocations

The Memory Debugger uses the bit pattern in this box when it paints newly deallocated heap memory. For more information, see "**Pattern for allocations**" on [page 183](#).

Apply pattern to deallocations

When **On** is selected, the Memory Debugger paints deallocated memory using the bit pattern shown in the **Pattern for deallocations** text field.

Hoarding

The Memory Debugger can delay handing freed memory back to the heap manager. This is called *hoarding*. For more information, see "**Hoarding**" on [page 32](#).

Figure 95: Memory Hoarding Area

Here is a description of these controls:

Hoard memory on deallocation

When **On** is selected, the Memory Debugger hoards memory. You can change this value while your program is executing.

If you set this value to **Off** while your program is executing, the Memory Debugger no longer hoards newly deallocated blocks. It does not, however, release blocks that it previously retained.

If the hoard is full and the Memory Debugger needs to hoard a new block, it releases the oldest blocks (that is, those that it first hoarded) so there's enough room in

its hoard buffer. You can change the size of the hoard using the next two controls.

Maximum KB to hoard

By default, the hoard can grow to 256 KB. You can change the hoard's buffer size by changing this value.

Maximum blocks to hoard

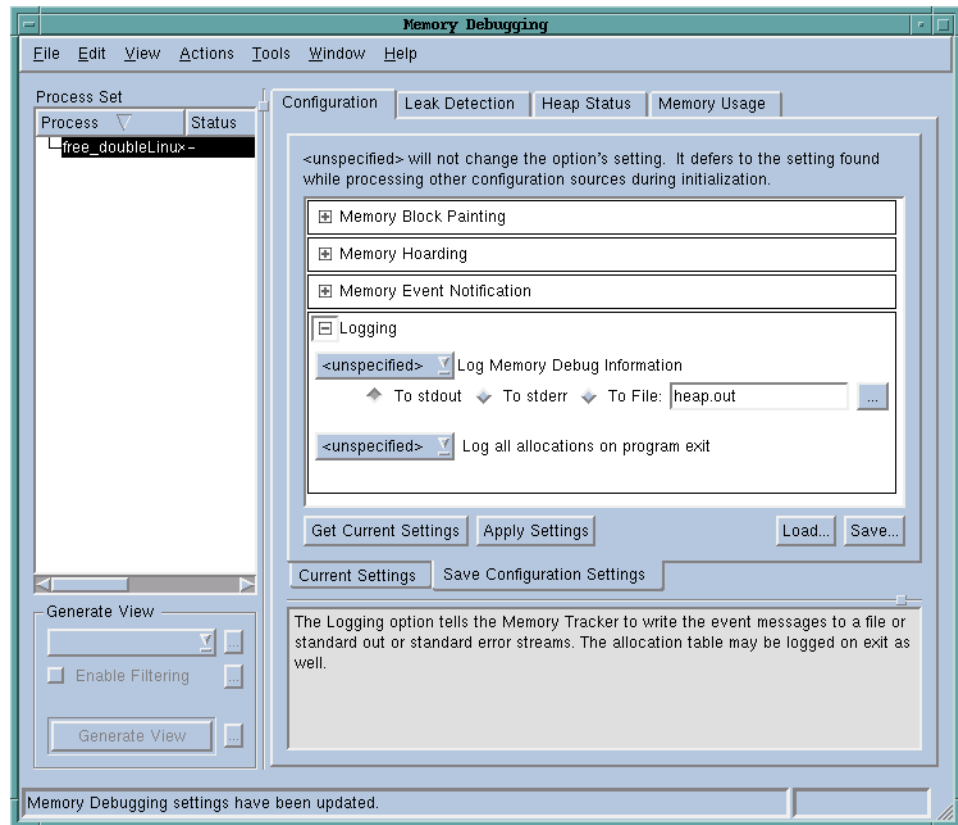
By default, the hoard can contain up to 32 memory blocks. You can change the number of blocks by changing this value.

The gray area underneath these controls indicates the **Current Size** of the hoard. You are told how many kilobytes the hoard is using and how many different blocks are contained within it.

Save Configuration Page


The Save Configuration Page contains four sets of controls. The first three, **Event Notification**, **Block Painting**, and **Hoarding** are the same as the controls within the Current Settings Page and have already been discussed in this topic. The fourth, **Logging**, is new.

Figure 96: Configuration Page



The controls in this area are:

Log Memory Debug Information

When set to **On**, the Memory Debugger writes its information to stdout, stderr, or to a file. You can edit the file name. Select the  button to name the directory into which the Memory Debugger writes information.

By default, it writes information into the program's directory.

Log all allocations on exit

When set to **On**, the Memory Debugger writes allocation information to the location set in the **Log Memory Debug Information** command.

The four commands at the bottom are as follows:

Get Current Settings

Sets the controls within this page to be the same as those that set on the **Current Settings** Page.

Apply Settings

Sets the controls on the **Current Settings Page** to be the same as those on this page. That is, consider this command to be the opposite of **Get Current Settings**.



This command ignores changes that occur within the Logging area. Logging can only be enabled if it is enabled in a default.hiarc file contained in your current directory or in your .totalview/hia directory. If your configuration file has another name or is stored elsewhere, you must name the file's name and location in the TVHEAP_ARGS variable.

Load

Reads a saved configuration file and sets the controls on this page to the values with the saved configuration file. After loading configurations, you still need to use the **Apply Settings** command to make them active.

After pressing this button, the Memory Debugger displays an explorer window that you can use to locate the file you want to load.

Save

Writes the configuration displayed in this page to a file. After pressing this button, the Memory Debugger displays an explorer window that you can use to locate the directory into which you want to write the file. You can also use the explorer window to enter a name for this file

For other information on this window, see:

- "Process Set Selection" on page 175
- "Generate View Area" on page 175
- "Block Painting" on page 182
- "Hoarding" on page 184
- "Event Notification" on page 181
- "Leak Detection Page" on page 187
- "Heap Status Page" on page 191
- "Memory Usage Page" on page 195

Leak Detection Page

The Memory Debugger can display information about the leaks it discovers in two ways: using a Source View or a Backtrace View. Each view displays approximately the same information.



Be careful how many processes you select. With large multiprocess programs, you might be asking the Memory Debugger to process and analyze an enormous amount of data. In most cases, if you select one or two significant processes, you'll receive the information you need. Although the process of generating a view is lengthy, you can redisplay the information quickly after the Memory Debugger creates it.

Source View

The Source View organizes the leaks in your program by the program, routine, file, and block.

To create this view:

- Select the processes for which you want information in the **Process Set** area.
- Select **Source View**, and then select **Generate View**.

In this view, the first column, **Process**, contains a hierarchical display organizing your program's information. The Backtrace and Source Panes contain additional information about the line you select in the Memory Blocks Pane. In other words, this view organizes the information in the same way that your program is organized.

Figure 97 on page 188 shows a Source View. In this figure, the bottom-most rows in the hierarchy contain information about an individual leak. As you go up the tree towards the process name, the Memory Debugger summarizes the number of bytes and the number of leaks associated with the information at lower levels of the tree. In this example, the program leaked 625.23 KB and 3,140 allocations were associated with leaks.



This explanation and the figure underemphasize the leak summary. Programs do leak memory. it is usually not practical to fix all leaks. If you click on the Bytes columns, the Memory Debugger sorts the table so that you can see what locations are leaking the most memory. This lets you focus on places leaking the most memory.

When you click on a line in the Memory Blocks Pane, the Memory Debugger shows information in the Backtrace Pane, as follows:

- The backtrace being displayed is the one that existed when your program allocated the memory block. The Memory Debugger highlights the frame that it thinks is the one you should be focusing on. That is, it highlights where the memory allocation was made. If it guesses wrong, you can reset the hierarchy of backtraces by right-clicking your mouse on the backtrace that you want displayed, as follows.
From the context menu, select **Set allocation focus level**.

For example, assume that you have created a function named **my_malloc()** that filters all of your memory allocations. The Memory Debugger would probably guess that this is the function to highlight in the Backtrace Pane. However, you probably want to set the allocation focus

Figure 97: Leak
Detection
Page: Source
View

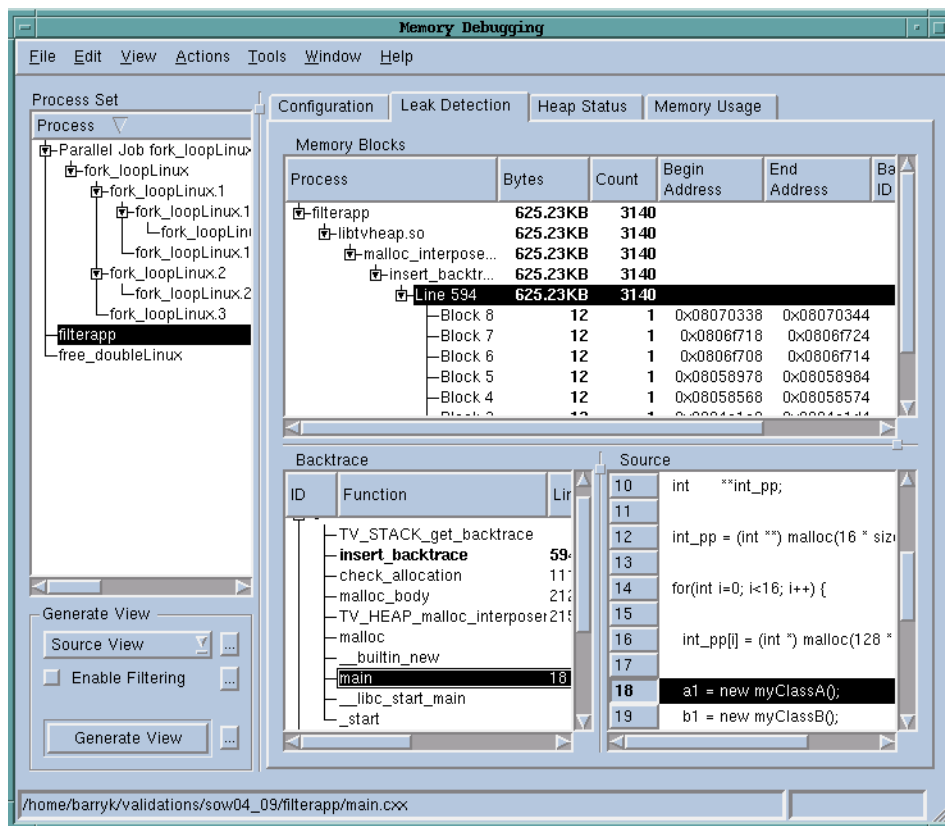
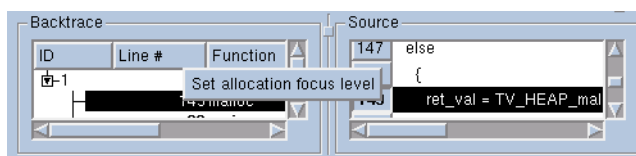


Figure 98: Backtrace and Source
Panels

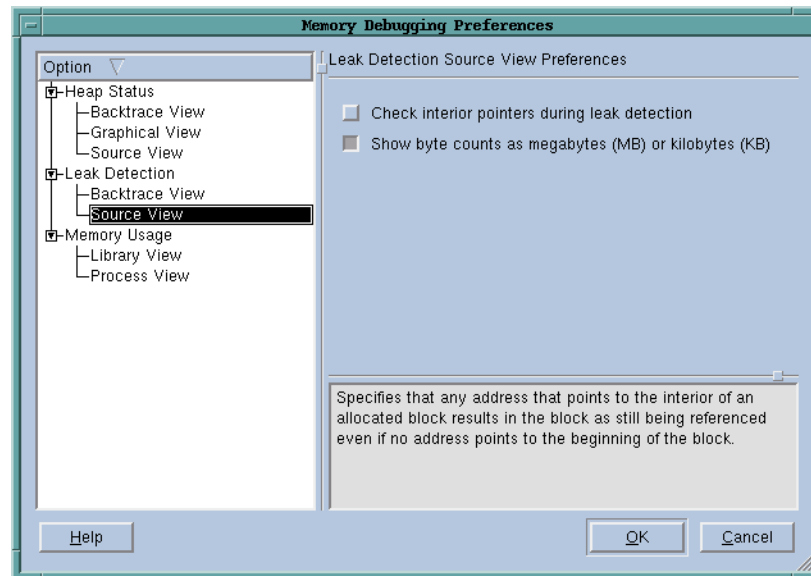



on the function that called **my_malloc()**. Do this by selecting that function, and then right-clicking on it to invoke the **Set allocation focus level** command.

- The Source Pane shows the line in your program that contained the memory allocation statement. When you click on a backtrace ID, the Memory Debugger updates the Source Pane to show the line. The line number associated with this line is the same line number that appears in the Process Window Source Pane.

You can set two preferences for Leak Detection views. After displaying the preferences dialog box, the Memory Debugger displays the following dialog box:

Figure 99: Leak Detection Source View Preferences



To set preferences associated with the Source View, select the  button within the Generate View area on the left. The preferences are as follows:

Check interior pointers during leak detection

Tells the Memory Debugger to consider a block as being referenced if a pointer is pointing anywhere within the block instead of just at the block's starting location. In most programs, the code should be keeping track of the block's boundary. However, if your C++ program is using multiple inheritance, you may be pointing into the middle of the block without knowing it.

Use this option with some caution as it can affect performance.

Show byte counts as megabytes (MB) or kilobytes (KB)

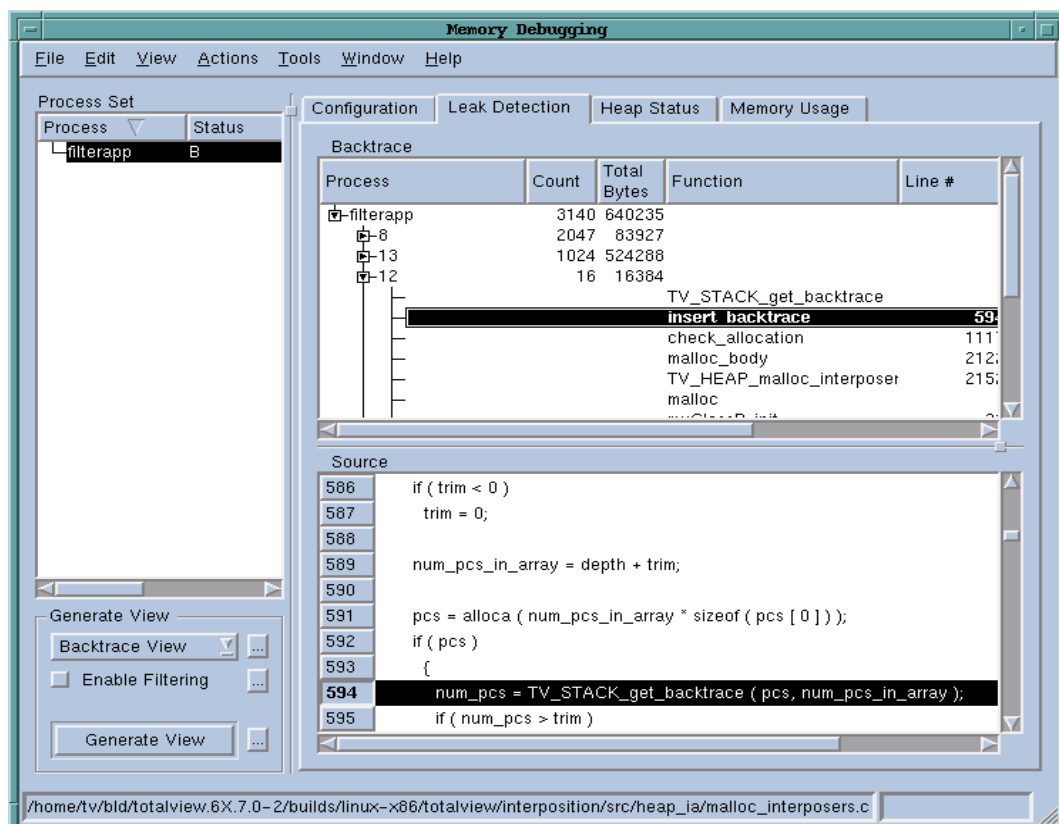
By default, the Memory Debugger displays memory sizes in KB. Selecting this check box tells the Memory Debugger to choose the most convenient size.

Backtrace View

The Backtrace View organizes the leaks in your program by the backtrace number created by the Memory Debugger. To create this view, select **Backtrace View**, and then select **Generate View**. In this view, the first column, **Process**, has a numeric list of all the backtrace ID numbers that the Memory Debugger creates.

When you look at one backtrace, you might be seeing the rolling together of many leaks into one. You can tell how many leaks are associated with

Figure 100: Leak Detection Page: Backtrace View

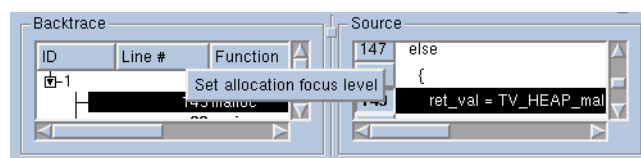


one ID by looking at the **Count** column. In this example, 16 leaks are associated with backtrace ID 12.

When you click on a line having a source code associated with it, the Memory Debugger displays that line in its Source Pane.

The backtrace being displayed is the one that existed when your program allocated the memory block. The Memory Debugger highlights the frame that it thinks is the one you should be focusing on. That is, it highlights where the memory allocation was made. If it guesses wrong, you can reset the hierarchy of backtraces by right-clicking your mouse on the back trace that you want displayed, as follows.


Figure 101: Backtrace and Source Panes



From the context menu, select **Set allocation focus level**.

For example, assume that you have created a function named **my_malloc()** that filters all of your memory allocations. The Memory Debugger would probably guess that this is the function to highlight in the Backtrace Pane.

However, you probably want to set the allocation focus on the function that called **my_malloc()**. Do this by selecting that function, and then right-clicking on it to invoke this command.

To set preferences associated with the Backtrace View, select the  button within the Generate View area on the left. The preferences are as follows:

Check interior pointers during leak detection

Tells the Memory Debugger to consider a block as being referenced if a pointer is pointing anywhere within the block instead of just at the block's starting location. In most programs, the code should be keeping track of the block's boundary. However, if your C++ program is using multiple inheritance, you may be pointing into the middle of the block without knowing it.

Use this option with some caution as it can affect performance.

Show byte counts as megabytes (MB) or kilobytes (KB)

By default, the Memory Debugger displays memory sizes in KB. Selecting this check box tells the Memory Debugger to choose the most convenient size.

Other topics that contain information are:

- **Configuration Page** on page 179
- **Heap Status Page** on page 191
- **Memory Usage Page** on page 195

Heap Status Page

The Heap Status Page displays information about all memory blocks that your program has not yet freed. The views shown in this page can be quite large. You can tell the Memory Debugger to display a Graphical, Source, or Backtrace View. Figure 102 on page 192 shows a Heap Status Source View.

Source and Backtrace Views

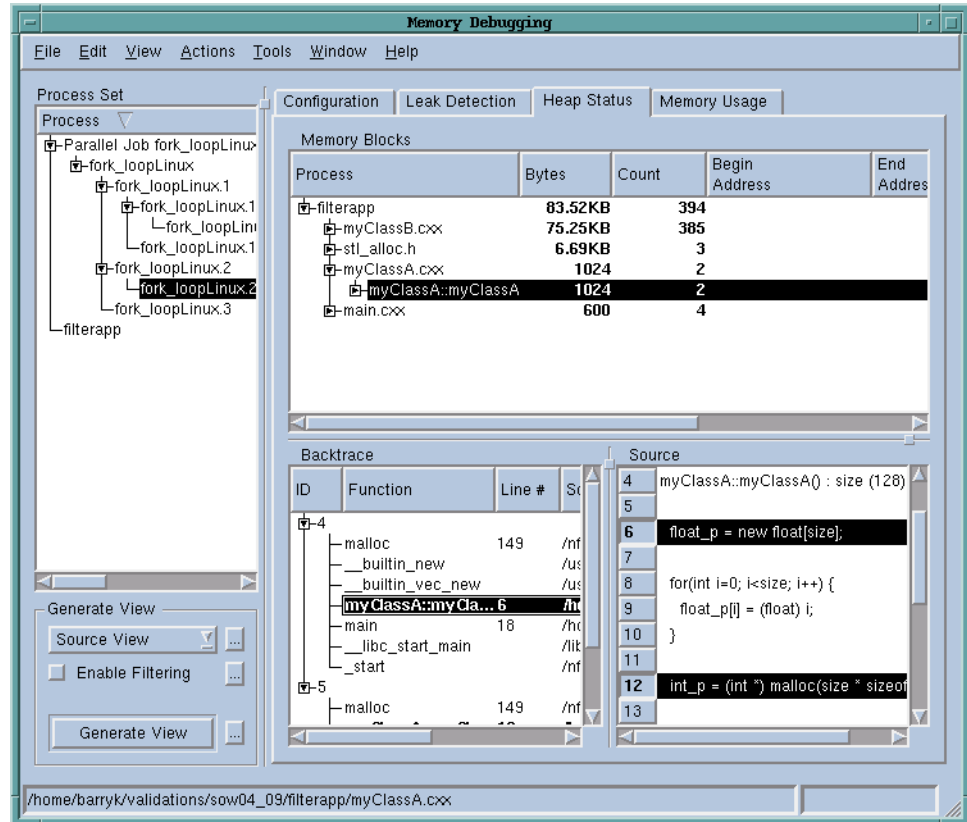
The Source and Backtrace Views within the Leak Detection page contain the same type of information that these view contain with the Heap Status Page. The sole difference is, of course, that these views in the Heap Status Page contain all memory allocations, not just allocations that represent leaks.

In most cases, an individual item is not very remarkable or noteworthy. However, the "rolled-up" information about your allocations can help you better understand your program's behavior.

For example, if your program's size is greater than you'd expect it to be, you can select the **Bytes** column so that the largest allocations are all grouped together. Concentrating on the statements allocating the most memory should lead you understand your program's behavior.

Similarly, if your program is allocating many small memory blocks, these allocations might be hurting performance. Looking at the information in

Figure 102: Heap Status Page: Source View



the **Bytes** and **Count** columns might also give you some hints about where you can improve performance.

You can also tell the Memory Debugger to display leaks in a different color. For more information, see “**Heap Status Preferences**” on page 194.

For more information on the contents of this page, see “**Leak Detection Page**” on page 187.

Graphical View

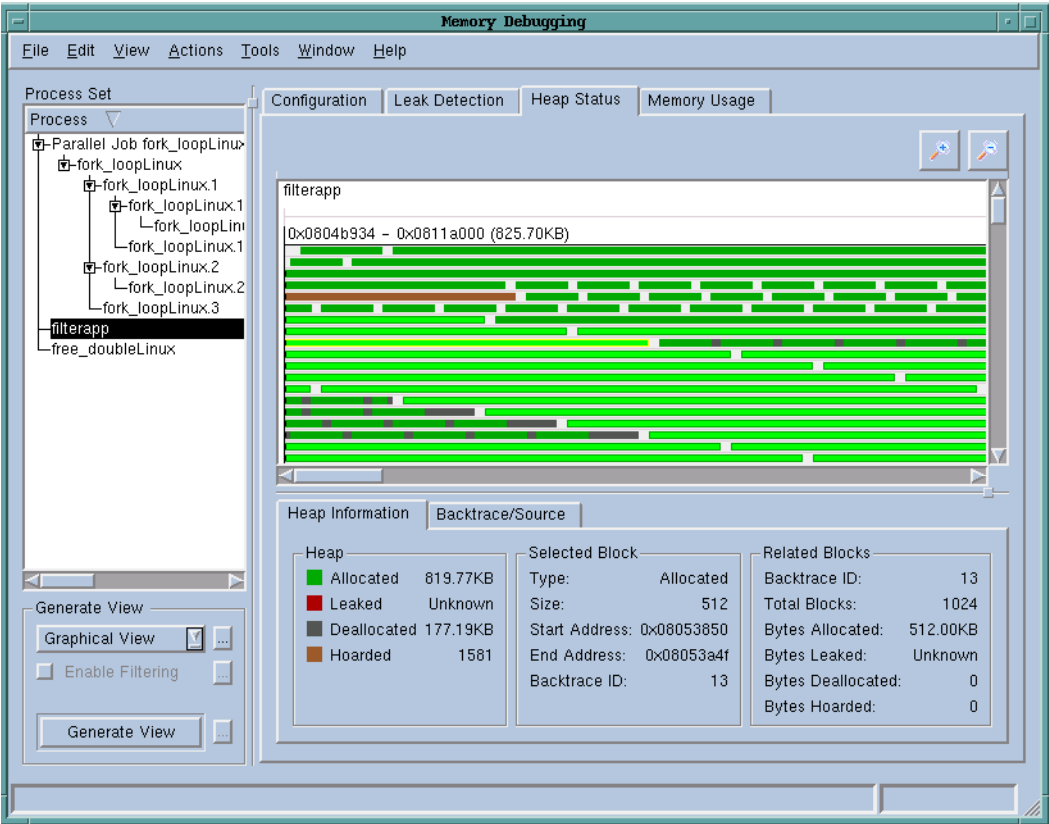
use of the heap API, the information presented within the Heap Status views can be overwhelming. In these cases and others, you may want to begin by displaying a graphical view of the heap. (See “**Heap Status Page: Graphical View**” on page 193)

You can create this view by selecting **Graphical View** and then pressing the **Generate View** button.

The Graphical View has two parts:

- The upper portion displays allocated blocks of memory.
- The bottom contains two tabs: **Heap Information** and **Backtrace/Source**. The information displayed when you select **Backtrace/Source** is the same as the Memory Debugger displays in the Source and Backtrace views. For information on the contents of these views, see “**Leak Detection Page**” on page 187

Figure 103: Heap Status Page: Graphical View



The length of each block in the upper portion is proportional to the size of the block. You can change the relative size of these blocks to see more or less information by selecting the magnifying glass icons above and to the right of the graphical display. The upper left corner within the graphical area contains general information.

The information in the top and bottom portions is linked. For example, if you select a block within the graphical area, the Memory Debugger displays information about the block in the bottom area. The Memory Debugger displays the selected block in yellow. It displays blocks having the same backtrace in green. If you are displaying the **Heap Information** page, you'll see summary information about this block. If you are displaying the **Source/Backtrace** page, you'll see the source line and backtrace associated with the block. If you select a source line or backtrace within this page, the Memory Debugger highlights the blocks associated with that source line and backtrace.

The three areas within the **Heap Information** page are as follows:

- **Heap:** Contains a key to the colors used in displaying blocks and a summary of how much memory is associated with each of the four allocation types displayed.
- **Selected Block:** Describes the block that you select. The only one of the five types whose meaning may be obscure is Backtrace ID. This is an identifier created by the Memory Debugger and is used to associated dif-

ferent backtraces together. You may find this number useful as you are examining memory information.

- **Related Blocks:** If the backtrace associated with a memory allocation is identical to the backtrace that existed when a previous allocation occurred, the Memory Debugger assigns the same backtrace ID to the newly created allocation. When you select a block, the Memory Debugger displays information about all blocks having the same backtrace ID.

Heap Status Preferences


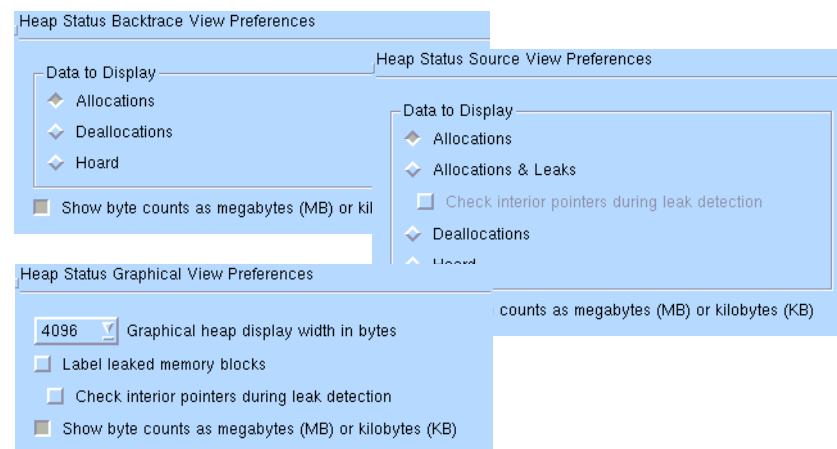
When you select the  button to the left of the view pulldown, the Memory Debugger displays a preference dialog box. The following figure shows the right side of each of the Heap Status preferences.

Figure 104: Heap Status Preferences:



Here is what these preferences let you do:

Data to Display (*Source and Backtrace View*) When displaying a Backtrace or Source View, tells the Memory Debugger to display allocations, deallocations, or hoarded information. In Source View, you can tell the Memory Debugger that it should also display leaked allocations.

Label Leaked Memory (*Graphical View*) Tells the Memory Debugger to display leaked memory in red.

Check interior pointers during leak detection.

(*Source and Graphical View*) Tells the Memory Debugger to consider a block as being referenced if a pointer is pointing anywhere within the block instead of just at the block's starting location. In most programs, the code should be keeping track of the block's boundary. However, if your C++ program is using multiple inheritance, you may be pointing into the middle of the block without knowing it.

Use this option with some caution as it can affect performance.

Graphical heap display width in bytes

(*Graphical View*) Defines how many bytes of block memory is displayed in each line within the graphical view.

Don't confuse this with the zoom controls. The zoom controls increase and decrease the size the Memory Debugger uses to display blocks. That is, it just changes how much is visible at one time.

Show byte counts as megabytes (MB) or kilobytes (KB)

(all views) When selected, the Memory Debugger chooses whether it should display memory in MB or KB. If this is not selected, the Memory Debugger always displays information in KB.

Other topics that contain information are:

- **Configuration Page** on page 179
- **Leak Detection Page** on page 187
- **Memory Usage Page** on page 195

Memory Usage Page

The Memory Usage Pages tells you how your program is using memory and where this memory is being used. One way to use this page is to compare memory use over time so that you can tell if your program is leaking memory. If a program is leaking memory, you'll see that the amount of memory being used steadily increases over time. You can also compare memory use between processes, which can tell you if a process is using more memory than you expect.



You do not need to enable memory debugging to obtain a Memory Usage View.

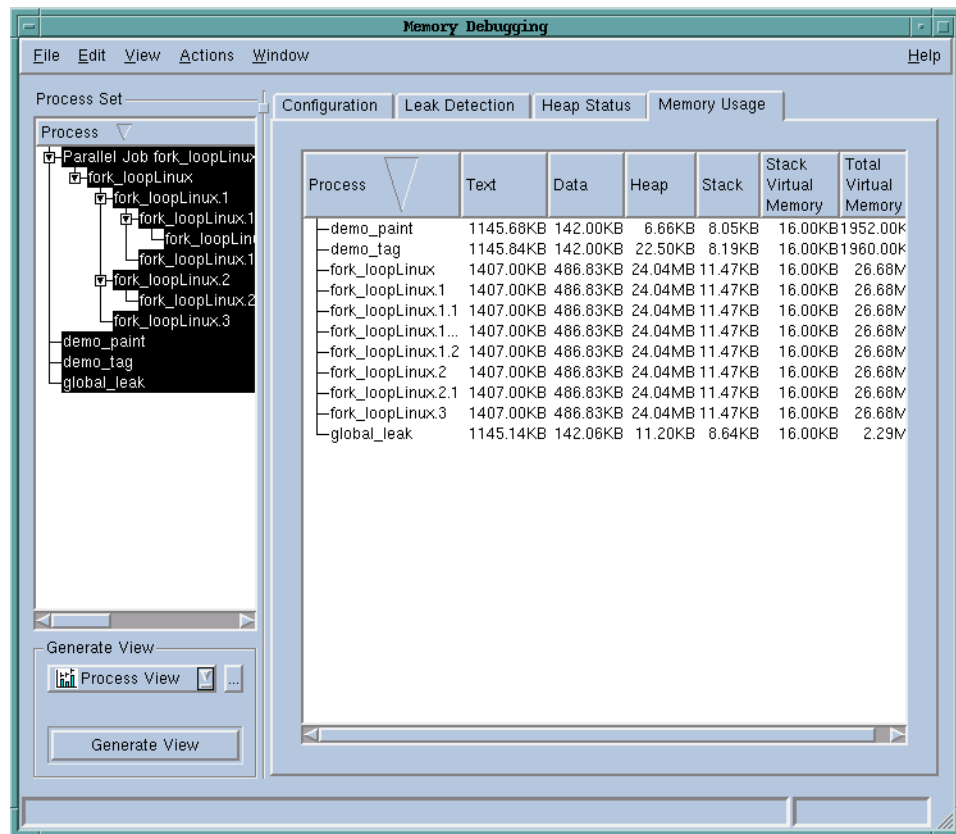
The Memory Debugger can present either a Process or Library View. Here is an example of the Process View.

Clicking on a column heading sorts the information from maximum to minimum or vice versa.

Notice that if you add the memory values of all columns but the last, the sum doesn't equal the last column's value. This are several reasons for this. For example, most operating systems divide segments into pages, and information in a segment does not cross page boundaries. Another reason is that a process could map a file or an anonymous region. Areas such as these are part of what you'll see in the Stack Virtual Memory column. However, they are not shown elsewhere.

The definitions for these columns are as follows:

Process	The name of your process.
Text	The amount of memory used for storing your program's machine code instructions.
Data	The amount of memory used for storing uninitialized and initialized data.
Heap	The amount of memory currently being used for data created at runtime.

Figure 105: Memory Usage
Page: Process View**Stack**

The amount of memory used by the currently executing routine and all the routines in its backtrace.

If you are looking at a multi-threaded process, Total-View only shows information for the main thread's stack. Note that the stack size of some threads do not change over time on some architectures.

On some systems, the space allocated for a thread is considered as being part of the heap.

Stack Virtual Memory

The logical size of the stack is the difference between the current value of the stack pointer and the value reported under the **Stack** column. This value can differ from the size of the virtual memory mapping in which the stack resides.

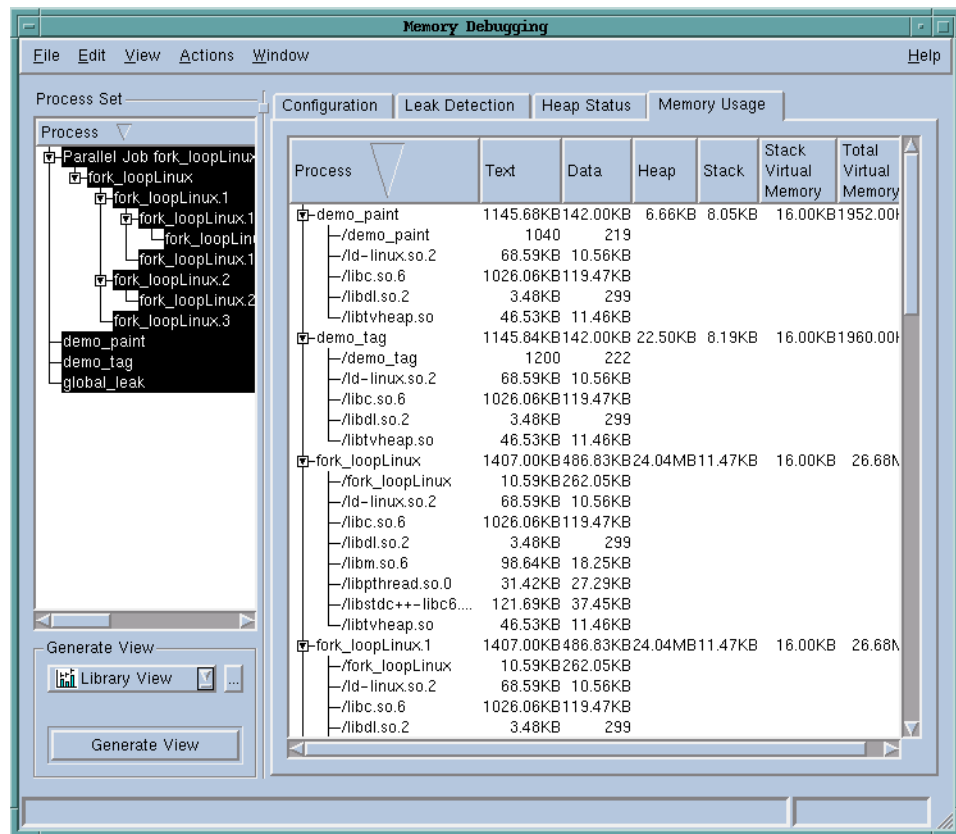
Total Virtual Memory

The sum of the sizes of the mappings in the process's address space.

The Library Pane shows which library files are contained within your executable. In addition to the same kind of information as was shown in the Process View, this view shows the amount of memory used by the text and data segments of these libraries. (See Figure 106 on page 197.)

Other topics that contain information are:

Figure 106: Memory Usage
View: Library View



- Configuration Page on page 179
- Leak Detection Page on page 187
- Heap Status Page on page 191

File Menu Commands

The following commands are on the File pulldown:

- File > Preferences on page 197
- File > Close on page 198

File > Preferences

Tells the Memory Debugger to open the **Preferences** Dialog Box. This is the same box that opens when you select the ... button that is next to the View pulldown on the right side of this window. For information on the contents of this window, see:

- Leak Detection Page on page 187
- Heap Status Page on page 191

File > Close

Tells the Memory Debugger to close this Memory Debugging window.

Edit Menu Commands

The following commands are on the File pulldown:

- **Edit > Copy** on page 198
- **Edit > Select All** on page 198
- **Edit > Find** on page 198
- **Edit > Find Again** on page 199

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

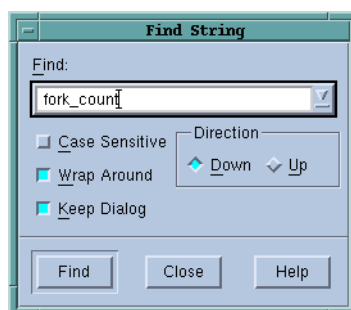
Edit > Select All

Selects and highlights all the elements in the current area.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 107: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. Be selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option. |
| Keep Dialog | If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box. |

<i>Direction</i>	Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file."
Find	Tells TotalView to search for the text within the Find box.
Close	Closes the Find dialog box.

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The following commands are on the View pulldown:

- **View > Collapse All** on page 199
- **View > Expand All** on page 199

View > Collapse All

Collapses all trees. That is, this is the equivalent to clicking ever – icon within the current area of the Memory Debugging Window.

View > Expand All

Expands all trees that are not displaying all of their information. That is, this is equivalent to clicking every + icon within the Memory Debugging Window.

Action Menu Commands

The following commands are on the **Action** pulldown:

- **Action > Generate View** on page 199
- **Action > View Preferences** on page 199

Action > Generate View

When any page except Configuration is being displayed, selecting this command displays a submenu. This submenu contains all the views that you can generate for that page. After making a selection, the Memory Debugger generates the view. For more information, see:

- **Leak Detection Page** on page 187
- **Heap Status Page** on page 191
- **Memory Usage Page** on page 195

Action > View Preferences

When any page except Configuration is being displayed, selecting this command displays a submenu. This submenu contains all the views that you can generate for that page. After making a selection, the Memory Debugger

displays the Preferences dialog box. The preferences for the selected view are displayed. For more information, see:


- **Leak Detection Page** on page 187
- **Heap Status Page** on page 191

Tools Menu Commands

The only command on the Tools pulldown is **Tools > Filters**.

Tools > Filters

The amount of information that the Memory Debugger displays when you ask for a Leak Detection or Heap Status View can be considerable. In addition, this information includes memory blocks allocated within any shared library used by your program. In other cases, your program may be allocating memory in many different ways and you only want to focus on a few of them. You can eliminate information from the display by using a filter. Filtering is a two-step process:

- 1 Create a filter by selecting the  button that is to the right of the **Enable Filtering** check box within the **Generate View** area. You can also use the **Tools > Filter** command.
- 2 At a later time, select the **Enable Filtering** check box.

When filtering is enabled, the Memory Debugger looks at each filter that you have created and enabled and applies it to the view's data. In addition, each can have any number of actions associated with it.

Adding, Deleting, Enabling and Disabling Filters


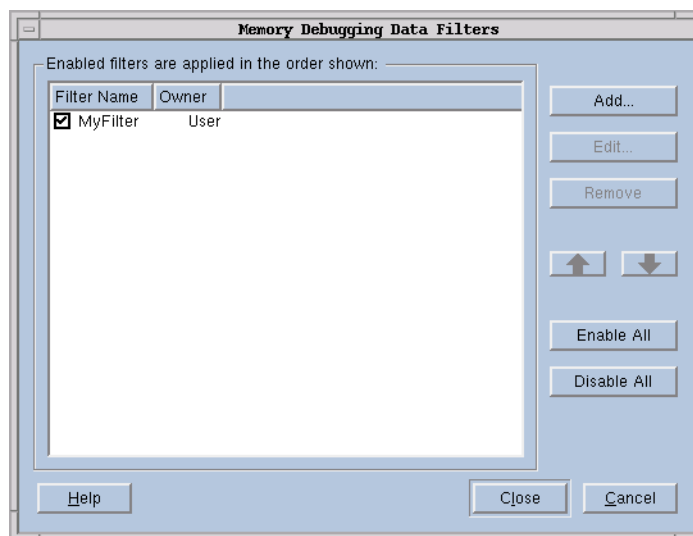
After you select the  button that is to the right of the **Enable Filtering** check box, the Memory Debugger displays a dialog box that allows you add, delete, enable, delete, and change the order in which the Memory Debugger applies filters.

Figure 108: Memory Debugging Data Filters Dialog Box



The controls within this dialog box are as follows:

☒ **Enable and Disable**

When checked, the filter is enabled.

Add

After pressing this button, the Memory Debugger displays the **Add Filter** dialog box. Using that dialog box, you can define one filter. That dialog box will be discussed later in this section.

Edit

Displays a dialog box that allows you to change the selected filter's definition. The displayed **Edit Filter** dialog box is identical to the **Add Filter** dialog box.

Remove

Deletes the selected filter.



Up and Down

Moves a filter up or down in the filter list. As the Memory Debugger applies filters in the order in which they appear in this list, you should place filters that remove the most entries at the top of the list. As filtering can be a time-consuming operation, this can increase performance.

Enable All

Enables (checks) all filters in the list.

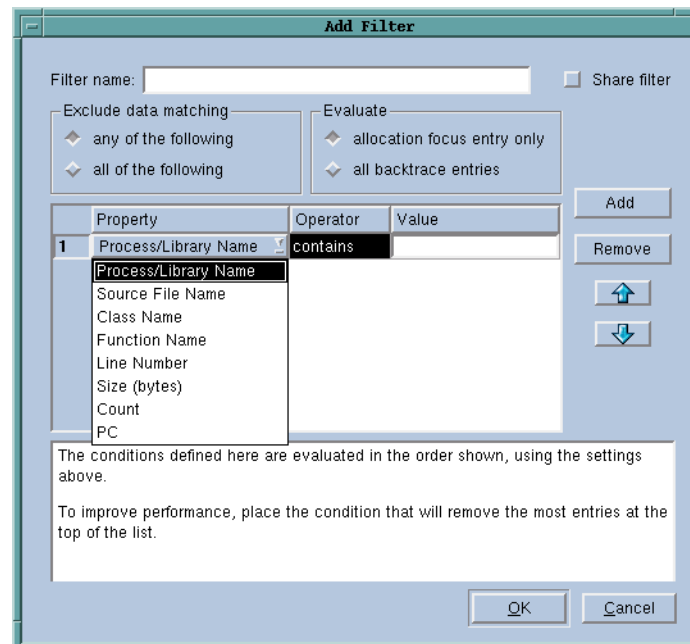
Disable All

Disables (unchecks) all filters in the list.

Adding and Editing Filters


After you select the **Add** button within the **Memory Debugging Data Filters** dialog box, the Memory Debugger displays the **Add Filter** dialog box:

Figure 109: Add Filter Dialog Box: Showing Properties



Selecting the **Edit** button within the **Memory Debugging Data Filters** dialog box tells the Memory Debugger to display a nearly identical window.

The controls within this window are as follows:

Filter name	Enter the name of the filter. This name will appear in the Memory Debugging Data Filters dialog box.
Share filter	Selecting this button tells the Memory Debugger that the filter you are creating will be shared. <i>Shared</i> means that anyone using TotalView can use the filter.
	<i>This button only appears if you have write permissions for the TotalView lib directory.</i>
Add	Pressing this button tells TotalView to add a blank line beneath the last criterion in the list. You can now enter information defining the criterion within this new line.
Remove	Deletes the selected criterion. To select a criterion, select the number to the left of the definition.
Up and Down	Changes the order in which criteria appear in the list. While changing the order doesn't change the results of the filtering operation, placing criteria that exclude the most information at the top of the list improves performance.
Exclude data matching	<p>If you have more than one criterion, the selected radio button indicates if <i>any</i> or <i>all</i> of the criteria have to be met.</p> <p>any of the following When selected, a memory entry is removed when the entry matches any of the criteria in the list.</p> <p>all of the following When selected, a memory entry is only removed if it fulfills all of the criteria.</p>
Evaluate	When evaluating a filter, you can limit which backtraces the Memory Debugger looks at.
allocation focus entry only	<p>When selected, tells the Memory Debugger that it should remove the entry only if the criteria you set is valid on the entry that is also the allocation focus.</p> <p>The allocation focus is the point in the backtrace where the Memory Debugger believes your code called malloc().</p> <p>For example, if you define a filter condition that says Function Name contains alloc1 and set this entry to allocation focus entry only, the Memory Debugger only removes blocks whose allocation focus contains alloc1. That is, it only removes blocks that were allocated directly from alloc1.</p>

In contrast, if you set this entry to **all backtrace entries**, the Memory Debugger removes all blocks that contain **alloc1** anywhere in their backtrace.

all backtrace entries

When selected, the Memory Debugger applies filter criteria to all function names within the backtrace..

Criteria

A filter is made up of criteria. Each criterion has three parts: a property, an operator, and a value. That is, you can indicate what the Memory Debugger looks for. For example, you can look for a Process/Library Name (the *property*) that contains (the *operator*) **strdup** (the *value*).

Property

When evaluating an entry, the Memory Debugger can look at one of eight properties for one criterion. These properties are shown in Figure 109 on page 201. Select one of the items from the pulldown list. These items are:

Process/Library Name
Source File Name
Class Name
Function Name
Line Number
Size (bytes)
Count
PC

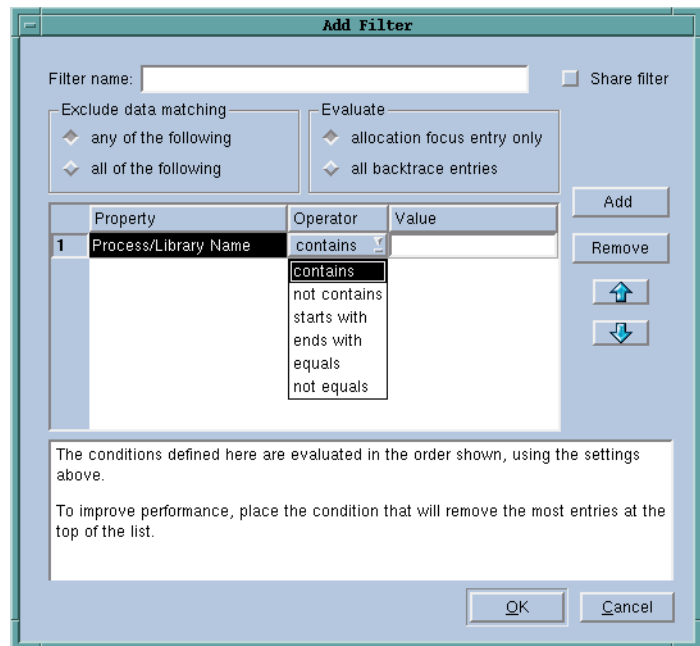
Operator

The operator indicates the relationship the *value* has to the *property*. These operators are shown in Figure 110 on page 204. Select one of the items from the pull-down list. If the item is a string, the Memory Debugger displays the following items:

contains
not contains
starts with
ends with

equals
not equals

Figure 110: Add Filter Dialog Box: Showing Operators



If the item is numeric, it displays the following list:

<=
<
=
!=
>
>=

Value Type a string that indicates what is being compared.

Window Menu Commands

The following commands are on the Window pulldown:

- **Window > Update** on page 204
- **Window > Update All** on page 205
- **Window > Duplicate** on page 205
- **Window > Root** on page 205

Window > Update

Updates the display of this window. *Update* means that the process is momentarily stopped so that TotalView can determine the program's state and the value of variables. It then updates this Variable Window and other

Variable Windows associated with this variable's process so that they contain updated values.

Window > Update All

Tells TotalView to update the contents of all windows. That is, TotalView fetches and then displays the current value of the information in all open windows. *Update* means that the process is momentarily stopped so that TotalView can determine the program's state and the value of variables. It then updates this Variable Window and other Variable Windows associated with this variable's process so that they contain updated values.

Window > Duplicate

Tells TotalView to create an identical copy of this Variable Window.

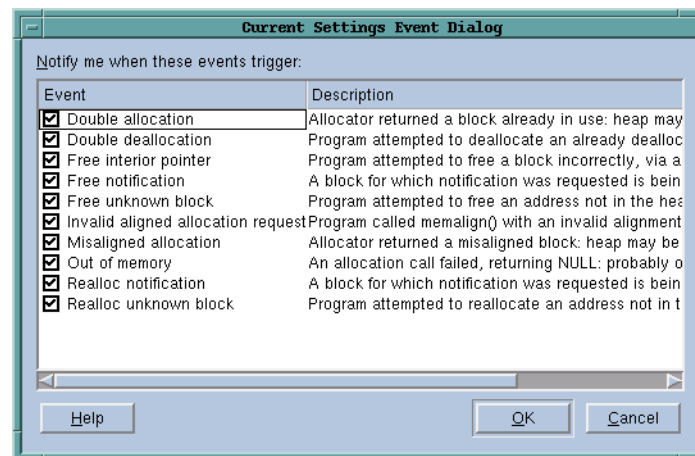
Window > Root

Tells TotalView to bring the Root Window to the top of the display.

Event Types

After selecting the Advanced button in the Configuration Page, the Memory Debugger displays the following dialog box.

Figure 111: Event Types Dialog Box



By default, you'll receive notifications for all events.

If an event is selected, the Memory Debugger will stop execution and displays the **Tools > Memory Event Details** Window. This window contains information about the memory block and why execution stopped.

If an event is not selected, you are not notified that an event occurred. Your settings here do not change how the Memory Debugger tracks allocations and deallocations. These settings only indicate if you are notified when an event occurs. For example, if no events are selected, the Memory Debugger still tracks all allocations and deallocations.

Thread Objects Window



This help file contains information for the Thread Objects Window. As the information TotalView displays is specific to your operating system, choose the one for which you want information:

- HP Tru64 UNIX
- IBM AIX on page 210

HP Tru64 UNIX

After you select the **Tools > Thread Object** command, TotalView displays a window containing the following two tabs:


- *Mutexes Page*
- **Condition Variables** on page 209

Mutexes Page

Displays a list of all of a process's mutexes. A *mutex* is a mutual exclusion object that allows multiple threads to synchronize access to shared resources. A mutex has two states: *locked* and *unlocked*. Once a mutex is locked by a thread, other threads attempting to lock it will block. Only after a locking thread unlocks (releases) the mutex can one of the blocked threads acquire (lock) the mutex and proceed.

For each mutex, TotalView displays the following information:


ID	The sequence number that the thread package assigns to a mutex. Diving into this field opens a Variable Window containing a view of the mutex's data.
Type	<p>The mutex type. These types are set using the pthread_mutexattr_settype() call on the attribute object before the mutex is initialized.</p> <p>This is a mutex type number and a single-character abbreviation of the type name. (While your system may have other types available, TotalView only shows these three types.)</p>

N	A normal mutex.
R	A recursive mutex.
E	An error-check mutex. Error-check mutexes contain additional information for use in debugging, such as the thread ID of the locking thread. During program development, you should use error-check mutexes in place of normal mutexes, and only switch to the simpler version when performance becomes an issue.
Flags	<p>This column contains hex strings that describe the current mutex flags and a one-character abbreviation for some flags:</p> <p>0x8 (M): Metered. The mutex contains metering information.</p> <p>0x4 (W): Waiters. One or more threads are waiting for this mutex. By default, waiting threads are shown in red. Their color is the same as the thread's error state flag color.</p> <p>0x2 (P): Locked. The mutex is locked. By default, locked mutexes are shown in blue; their color is the same as the thread's stopped state flag color.</p> <p>0x1 (N): Name. This mutex has a name.</p> <p>While your system may use additional flag bits, TotalView only shows names for these flags.</p>
Owner	<p>If the mutex is locked, this field displays the locking thread's system thread ID (TID). This TID is only available for error-check mutexes.</p> <p>Diving or selecting on this number tells TotalView to display the locking thread's Process Window. This is the same window that TotalView would display if you dive or select the thread's entry in the Root Window's Attached Page.</p> <p>If threads are waiting for this mutex, their system TIDs are shown in the owner field, with one thread ID displayed on each line. You can open a Process Window for these waiting threads by diving or clicking on its number.</p>
	<i>If TotalView cannot obtain this information, it does not show blocked thread lines.</i>
Address	This field contains the memory address of the mutex. You can open a Variable Window containing a view of the mutex's data by diving on this field.
Name	If the mutex has a name, it is shown here. If you are using version 4.0D or later of the operating system, the pthread_mutex_setname_np() routine provides the mutex's name. However, this routine is not portable.

Condition Variables

The Condition Variables Page lists all the condition variables known in this process.

For each condition variable, TotalView displays the following information:

ID	The ID is the sequence number assigned to this condition variable by the threads package. Diving into this field opens a Variable Window containing a view of the condition variable's data.
Flags	<p>The information in this column is a hex string containing the current condition variable's flags and a one-character abbreviation for some of the flags:</p> <p>0x8 (M): Metered. This condition variable contains metering information.</p> <p>0x4 (W): Waiters. One or more threads are waiting for this condition variable. By default, this is shown in red, which is the same as the thread's error state flag color.</p> <p>0x2 (P): Pending. A wakeup is pending for this condition variable. By default, this is shown in blue; its color is the same as the thread's stopped state flag color.</p> <p>0x1 (N): Name. The condition variable has a name.</p> <p>While your system may use more flags, TotalView only shows these four flag names.</p>
Waiters	If threads are waiting for this condition variable, the debugger displays their system thread IDs (TIDs), one thread for each line, on the lines following the condition variable. Diving or selecting entries in the list of waiting threads opens windows for them.
 Mutex	<p>If TotalView cannot obtain this information, it does not show waiting threads.</p> <p>This field contains the ID of the mutex that guards the condition variable. If TotalView can translate the ID into an address, diving into this field opens a Variable Window containing a view of the guard mutex's data.</p> <p>TotalView can only translate this ID if it has already been initialized. That can be done statically or by using an attributes object. See the <code>pthread_cond_init(3)</code> and <code>pthread_mutex_init(3)</code> man pages for more information.</p>
Address	This field has the condition variable's memory address. Diving into the address field opens a Variable Window containing a view of the actual condition variable's data.
Name	If the condition variable has a name, it is shown here. If you are using version 4.0D or later of the operating system, the <code>pthread_mutex_setname_np()</code> routine provides the condition variable's name. This routine is not portable.

IBM AIX

After you select the **Tools > Thread Object** command, TotalView displays a window containing the following four pages:

- **Mutexes Page** on page 210
- **Condition Variables Page** on page 211
- **R/W Locks Page** on page 212
- **Data Keys Page** on page 214

You must set up to six variables when debugging threaded applications. Here's what you would do in the C shell:

```
setenv AIXTHREAD_MNRATIO "1:1"
setenv AIXTHREAD_SLPRATIO "1:1"
setenv AIXTHREAD_SCOPE "S"
setenv AIXTHREAD_COND_DEBUG "ON"
setenv AIXTHREAD_MUTEX_DEBUG "ON"
setenv AIXTHREAD_RWLOCK_DEBUG "ON"
```

The first three variables must be set. Depending upon what you need to examine, you will also need to set one or more of the "DEBUG" variables.

Do not, however, set the **AIXTHREAD_DEBUG** variable. If you have set it, you should unset it before running TotalView



Setting these variables can slow down your application's performance. None of them should be set when you are running non-debugging versions of your program.

Mutexes Page

A mutex is a mutual exclusion object that allows multiple threads to synchronize access to shared resources. A mutex has two states: *locked* and *unlocked*. Once a mutex is locked by a thread, other threads attempting to lock it will block. Only after a locking thread unlocks (releases) the mutex can one of the blocked threads acquire (lock) the mutex and proceed.

This page contains a list of all mutexes known in a process.

For each mutex, TotalView displays the following information:

ID	The sequence number assigned to a mutex by the threads package. Diving into this field opens a Variable Window containing a view of the mutex's data.
Type	The mutex type. These types are set using the pthread_mutexattr_settype() call on the attribute object before the mutex is initialized. The type is one of the following:
Normal	A normal mutex.
Recurs	A recursive mutex.
ErrChk	An error-check mutex.
NRecNP	A non-portable, non-recursive mutex.
RcurNP	A non-portable, recursive mutex.
FastNP	A non-portable, fast mutex.

State	The mutex lock state is displayed as follows:
Unlocked	The mutex is unlocked.
Locked	The mutex is locked. By default, this is shown in blue; its color is the same as the thread's stopped state flag color.
Pshared	This value indicates if the mutex can be shared by other processes.
Private	The mutex can only be manipulated by threads in the process that initialized the mutex.
Shared	The mutex can be manipulated by any process that has access to the mutex's memory. (Some versions of IBM's system libraries cannot provide information on shared mutexes. If this information is not available, TotalView only describes private mutexes.)
Owner	<p>If the mutex is locked, this field displays the locking thread's system TID.</p> <p>Diving on this number tells TotalView to display the locking thread's Process Window. This is the same window that TotalView would display if you dive or select the thread's entry in the Root Window's Attached Page.</p> <p>If threads are waiting for this mutex, their system TIDs are shown beneath the owner field, with one thread ID displayed on each line. You can open a Process Window for these waiting threads by diving or selecting on its number.</p> <p><i>If TotalView cannot obtain this information, it does not show waiting thread system TIDs.</i></p>
Address	This field contains the memory address of the mutex. You can open a Variable Window containing a view of the mutex's data by diving on this field.

Condition Variables Page

The Condition Variable Page lists all the condition variables known in this process.

For each condition variable, TotalView displays the following information:

ID	The ID is the sequence number assigned to this condition variable by the threads package. Diving into this field opens a Variable Window containing a view of the condition variable's data.
Pshared	This value indicates if the condition variable can be shared by other processes.
Private	The condition variable can only be manipulated by the process that initialized it.
Shared	The condition variable can be manipulated by any process that has access to its memory. (Some versions of IBM's system libraries cannot provide information on



	shared condition values to TotalView. If this information is not available, TotalView only describes private condition values.)
Waiters	If threads are waiting for this condition variable, the debugger displays their system TIDs, one thread for each line, on the lines following the condition variable. Diving or selecting entries in the list of waiting threads opens windows for them.
	<i>If TotalView cannot obtain this information, it does not show waiting threads.</i>
Mutex	<p>This field contains the ID of the mutex that guards the condition variable. If TotalView can translate the ID into an address, diving into this field opens a Variable Window containing a view of the guard mutex's data.</p> <p>TotalView can only translate this ID if it has already been initialized. That can be done statically or by using an attributes object. See the following mutex and condition variable man pages for more information: pthread_cond_init (3), pthread_mutex_init (3), pthread_cond_initializer (3), and pthread_mutex_initializer (3).</p>
Address	This field has the condition variable's memory address. Diving into the address field opens a Variable Window containing a view of the actual condition variable's data.

R/W Locks Page

A read-write lock is a mutual exclusion object that allows multiple threads to synchronize access to shared resources. A read-write lock has three states:

- Free
- Read-locked
- Write-locked

A free lock can be locked by any number of readers or by one writer. Once a read-write lock is locked by a thread for one kind of access, other threads attempting to lock it for other kinds of access will block. When locking threads unlock (release) the read-write lock, blocked threads can acquire (lock) it and proceed.

This page lists all read-write locks known in this process.

For each lock, TotalView displays the following information:

ID	This field contains the sequence number assigned to this read-write lock by the threads package. Diving into this field opens a window containing the read-write lock data.
-----------	---

State	This field displays the read-write lock state as follows:
Free	Unlocked.
Read	Locked for reading. By default, this is shown in blue; its color is the same as the thread's stopped state flag color.
Write	Locked for writing. By default, this is shown in blue; its color is the same as the thread's stopped state flag color.
Pshared	This value indicates if the read-write lock can be shared by other processes.
Private	The read-write lock can only be manipulated by the process that initialized it.
Shared	The read-write lock can be manipulated by any process that has access to its memory. (Some versions of IBM's system libraries cannot provide information on shared read-write locks to TotalView. If this information is not available, TotalView only describes private read-write locks.)
Owner	<p>If the read-write lock is locked, this field displays the system TID of a locking thread. Diving or selecting on this number tells TotalView to display the Process Window for that thread. TotalView displays the same window if you dive or select the thread's entry in the Root Window's Attached Page.</p> <p>If threads are waiting for this read-write lock, their system TIDs are shown beneath the system TID in this field, with one thread ID being displayed for each line in the window. That is, threads that are waiting to read and threads waiting to write are grouped together.</p> <p>You can open a Process Window for a waiting thread by diving or selecting its number.</p> <p>If TotalView cannot obtain this information, it does not show blocked thread lines.</p> <p><i>Some versions of IBM's system libraries cannot provide the correct owner TID for read-write locks locked for reading. In these cases, the owner TID can only be trusted when the lock is in its write state.</i></p>
Address	The memory address of the read-write lock. You can open a window displaying the read-write lock data by diving on this field.



Data Keys Page

A pthread-specific data key is an object that can have a pointer value of type `void *` associated with it for each pthread in a process.

This window contains a list of all keys known in this process.

TotalView displays information for each key. Many applications initially set keys to zero (the NULL pointer value) using `pthread_set_specific()`. Note that a key's information is not displayed until a thread sets a value for it, even if the value set is NULL.

ID	This field contains the sequence number assigned to this key by the threads package. Only the line for the first thread's value for a key will contain an ID; subsequent lines for the same key omit the ID as a way of visually grouping values with the same ID.
Thread	This field has the system TIDs of the threads that have a value for this key. Diving or selecting on this number tells TotalView to display the Process Window for the thread. TotalView displays the same window if you dive or select the thread's entry in the Root Window's Attached Page.
Value	This field contains the contents of the key for a pthread. Diving into this field opens a window containing a view of the actual key data.

File Menu Commands

The commands on the **File** pulldown are:

- **File > Close Similar** on page 214
- **File > Close** on page 214
- **File > Exit** on page 214

File > Close Similar

Closes this window and closes windows whose contents are similar to this window.

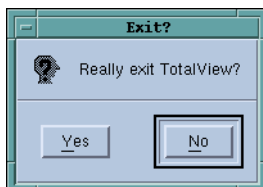
File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

Figure 112: File > Exit Dialog Box



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 215
- **Edit > Cut** on page 215
- **Edit > Copy** on page 215
- **Edit > Paste** on page 215
- **Edit > Delete** on page 215
- **Edit > Find** on page 216
- **Edit > Find Again** on page 216

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

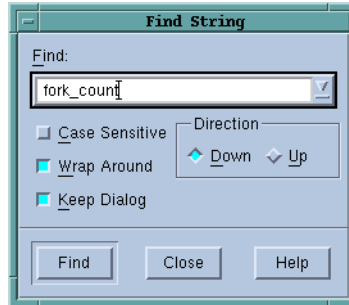
A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the

text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 113: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. By selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option. |
| Keep Dialog | If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box. |
| Direction | Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file." |
| Find | Tells TotalView to search for the text within the Find box. |
| Close | Closes the Find dialog box. |

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The commands on the **View** Pulldown are:

- **View > Dive ...** on page 217
- **View > Dive ... in New Window** on page 217

View > Dive ...

Tells TotalView to “dive” into the selected item. In all cases, “dive” means that TotalView will show more information. For example:

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

In both cases, if a Process or Variable Window already exists, TotalView brings that window to the top of the display.

Note that the Data Keys Page also has a **View > Dive Thread** command. This command tells TotalView to show a Process Window containing this thread.

View > Dive ... in New Window

Tells TotalView to open a “dive” into the selected item. In all cases, “dive” means that TotalView will show more information.

- If the item is a process, TotalView opens a Process Window for it.
- If the selected item is a variable, TotalView opens a Variable Window containing information about that variable.

Even if a Process or Variable Window already exists for a process or variable, TotalView always creates a new window for this information.

Note that the Data Keys Page also has a **View > Dive Thread in New Window** command. This command differs from **View > Dive Thread** in that information TotalView will not reuse an existing Process Window if the process isn’t being displayed.

Window Menu Commands

The commands in the **Window** pulldown are:

- **Window > Update** on page 217
- **Window > Update All** on page 218
- **Window > Memorize** on page 218
- **Window > Memorize all** on page 218
- **Window > Root** on page 218

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program’s state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

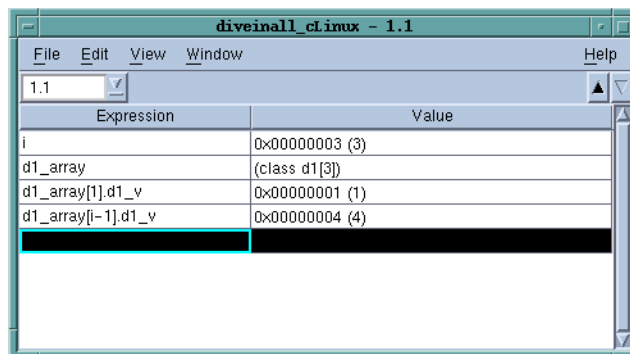
Expression List Window



Expression List Window Overview

Selecting the Process Window's **Tools > Expression List** command displays a window containing a list of variables and expressions. TotalView also displays this window after you select the **Add to Expression List** command from a context menu.

Figure 114: Tools > Expression List Window



This window can contain the values of as many variables and expressions so that you can monitor changes that occur as your program executes. That is, whenever your program halts in the thread listed in the **Threads** box, TotalView will reevaluate the value of everything in the **Expression** column.

Entering Variables and Expression into the Expression List Window

There are a number of ways to get information into the first column of this window:

- You can type information text into the bottom-most cell in the **Expression** column or edit text already entered in this column. The variable or expression that you enter is evaluated in the context of the current PC in the thread listed in the **Threads** box. For example, if you typed **my_var** into the window shown here, the value for it that TotalView

displays is the value of **my_var** in process 1, thread 1 and its scope is defined by the current PC. This means, for example, if you have two variables named **my_var**, TotalView will not change the context when the other **my_var** is in scope.

If you would like TotalView to change the scope in which it evaluates an expression, right-click on a row and select the **Compilation Scope > Floating** command. For more information, see “**View > Compilation Scope > Floating**” on page 132.

- Right click on something in the Process Window’s Source or Stack Frame Panes. From the displayed context menu, select **Add to Expression List**. Here’s is the context window that displays within the Process Window:

Figure 115: Context Menu



- Right click on something in the Variable Window. Select **Add to Expression List** from the displayed context menu. You can also use the **View > Add to Expression List** command.

When sending something to the **Expression List** Window, the cursor and your selection matter. If you click on a variable or select a row in the Variable Window, TotalView sends the variable to the **Expression List** Window. If you instead select some text with the Source or Stack Frame Panes, TotalView sends only that text. What’s the difference? In the figure at the beginning of this topic, notice that there are three different **d1_array** expressions.

- The first was added by just selecting part of what was displayed in the Source Pane.
- The second was added by selecting a row within the Variable Window.
- The third was added by clicking at a random point within the variable’s text in the Source Pane.

The scope that TotalView uses when looking up a variable when execution stops is the scope that existed when the variable was entered. If you want the scope to float so that a variable can be evaluated in different scopes, right-click within the variable’s row and select the **Compilation Scope > Floating** command. Selecting **Compilation Scope > Fixed** tells TotalView that it should only evaluate the variable in its original scope. For more information, see:

- **View > Compilation Scope > Fixed** on page 132.
- **View > Compilation Scope > Floating** on page 132.

Opening and Closing the Expression List Window

If you close the **Expression List Window** and then reopen it, TotalView remembers what you had previously entered. In other words, it doesn't delete what you enter. Instead, you must explicitly delete expressions using the **Edit > Delete Expression** and **Edit > Delete All Expressions** commands.

What Can You Enter in the Expression Column

You can type a variable or an expression within a cell in the **Expression** column or, if you select the **Add to Expression List** command, TotalView can add an entry. The expressions that enter are limited. They cannot contain function calls and they cannot create side-effects. A previous figure showed four different expressions.

<code>i</code>	A variable with one value. The Value column shows its value.
<code>d1_array</code>	An aggregate variable; that is, an array, a structure, a class, and so on. It's value cannot be displayed in one line. Consequently, TotalView just gives you some information about the variable. To see more information, dive on it. After diving, TotalView displays the variable in a Variable Window. Whenever you place an aggregate variable in the Expression column, you will need to dive on it to get more information.
<code>d1_array[1].d1_v</code>	This entity is an element within an array of structures. If TotalView can resolve what you enter in the Expression column into a single value, it will display a value in the Value column. If it can't, TotalView displays information in the same way that it displays information in the <code>d1_array</code> example.
<code>d1_array[i-1].d1_v</code>	This differs from the previous example in that the array index is an expression. Whenever execution stops in the current Thread, TotalView reevaluates <code>i</code> , so that the element within the array that is evaluated may change. The expressions you enter cannot include function calls.

Manipulating the Expression List Window

When you first bring up the **Expression List Window**, it contains two columns. However, TotalView can display additional columns. If you right click on a column headings, TotalView displays a context menu that shows your choices.

Here are operations you can perform by directly manipulating elements within the window:


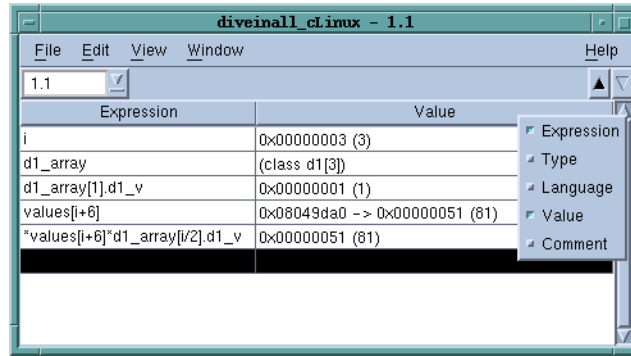
- The up and down arrows () on the right side of the toolbar allow you to change the order in which rows are displayed. For example, clicking on the down arrow moves the currently selected row one row lower in the display.
- You can change an expression by clicking within its text, then typing new characters and deleting others.

Figure 116: Tools > Expression List Window



- You can sort the contents of a column by clicking on the column header. For example, you could sort the **Value** column into an ascending order. After you click on the header, TotalView adds an indicator indicating that the column was sorted and the way in which it was sorted. Reclicking re-sorts the column into a descending order. Clicking a third time removes the results of sorting; that is, TotalView restores the window to what it was before you clicked on the column heading.
- Similarly, you can change a value in the **Value** column if that value is stored in memory. After entering a new value, TotalView replaces the old memory value with the one you just entered. This change cannot be undone.

Multiprocess/ Multithreaded Behavior

You can change the thread in which TotalView evaluates elements within the **Expressions** column by changing the value in the **Threads** box contained within the window's toolbar.

When you send a variable to the **Expression List** Window and a window is open, TotalView checks that window's thread. If it is the same as the thread associated with the Process or Variable Window from which you are sending the expression, TotalView adds it to the bottom of the list. If it is different, TotalView still adds it to the bottom of the list. It then duplicates the window and changes the thread of the newly created window. That is, adding a variable to the list can create a new window. In this example, each has the same list of expressions. The value of these expressions may differ as they show the values of the expressions within different threads.

In all cases, the list of expressions will always be the same. What differs is the context in which TotalView evaluates the window's expressions.

Similarly, if TotalView is displaying two or more **Expression List** Windows and you send from yet another process and thread combination, TotalView adds the variable to all of them, duplicates one of them, and then changes the duplicated window to this new combination.

File Menu Commands

The commands on the **File** pulldown are:

- **File > Preferences** on page 223
- **File > Save Pane** on page 223
- **File > Close Similar** on page 224
- **File > Close** on page 224
- **File > Exit** on page 224

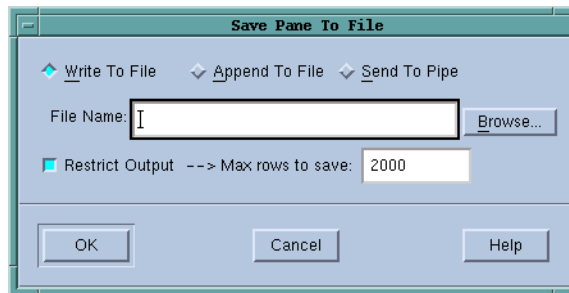
File > Preferences

Use this dialog box to set preferences for how TotalView will behave situations, as well as define some general characteristics. For more information, see “**File > Preferences**” on page 9, which is within the Root Windows help.

File > Save Pane

Use this dialog box to write the contents of the selected page, pane, or window.

Figure 117: File > Save Pane Dialog Box



Write to File

Tells TotalView to write information to a file. You can either enter the name of the file in the **File Name** field or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView overwrites it. If the file does not exist, TotalView creates the file before writing this information.

Append To File

Tells TotalView to add information to a file. You can either enter the name of the file in the **File Name** edit box or use the **Browse** button to move through the file system to select an existing file.

If the file already exists, TotalView adds this information to the end of the file. If the file does not exist, TotalView creates the file before writing this information.

Send To Pipe

Sends the data to the program or script named in the **File Name** field.

Restrict Output --> Max rows to save

If checked, TotalView will limit how much information it should send. If the default value of 2000 rows is not what you want, you can specify how many rows TotalView should write.

Edit Menu Commands

File > Close Similar

Closes this window and closes windows whose contents are similar to this window.

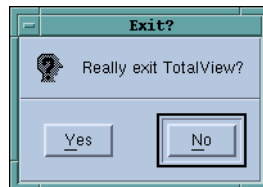
File > Close

Closes this window. No other windows are affected by this command.

File > Exit

Exits from TotalView. Before TotalView exits, it asks you to confirm that you really want to exit.

Figure 118: File > Exit Dialog Box



As TotalView exits, it kills all processes that it started. It does not, however, kill processes that were already running when you attached to them.

If TotalView has been told to automatically save action points, TotalView writes them to disk at this time. For more information, see the Root or Process Window's **File > Preferences** dialog box.

Edit Menu Commands

The commands on the **Edit** pulldown are:

- **Edit > Undo** on page 224
- **Edit > Reset Default** on page 224
- **Edit > Cut** on page 224
- **Edit > Copy** on page 225
- **Edit > Paste** on page 225
- **Edit > Delete** on page 225
- **Edit > Delete Expression** on page 225
- **Edit > Delete All Expressions** on page 225
- **Edit > Find** on page 226
- **Edit > Find Again** on page 226

Edit > Undo

While you are editing a variable's value or a data type declaration, this command restores the field to its original value.

Use the command to restore a value that you are currently editing. After the editing change is entered, you cannot undo your edit.

Edit > Reset Default

If you have made changes within the **Expression** or **Type** field, selecting this command removes all changes you had made so that what is displayed is what was entered or existed originally in window.

Edit > Cut

Copies the current selection to the clipboard and then deletes it. You can only use this command if the text being cut is editable. For example, you

cannot cut a programming language statement within the Process Window's Source Pane. You can, however, copy a programming language statement contained within the Source Pane (as well as data from other panes) after it is selected.

Edit > Copy

Copies the current selection to the clipboard. TotalView allows you to copy information from most of its windows and panes to the clipboard. However, you need to use Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Paste

Pastes the contents of the clipboard at the cursor location. Note that many places in many windows are not editable.

If information within TotalView is highlighted, the pasted information replaces the highlighted information. That is, the old information is deleted.

TotalView allows you to copy information in all of its windows and panes to the clipboard. However, you need to use the Ctrl+C and Ctrl+V accelerators to copy and paste information within dialog boxes.

Edit > Delete

Deletes the selected information. This deleted information is not placed onto the clipboard.

You can only delete information contained within editable areas.

A faster way to delete text is to select the text you want deleted and press the Delete key. If you are entering something else in a field, just select the text and then type your new information. TotalView will delete the selected information immediately before it inserts what you type or what you paste.

Edit > Delete Expression

Removes the currently selected row from the list of displayed. You cannot undo this operation.

Edit > Delete All Expressions

Removes all rows from the Expression List window. You cannot undo this operation.

Edit > Duplicate Expression

Makes a copy of the currently selected row and pastes it into the bottom row of the Expression List Window. TotalView duplicates the entire row.

This is useful when you want to track a group of similarly named variables at the same time. For example, you might duplicate

`my_array_var[random_index]` so that TotalView is also tracking `my_array_var[random_index+10]`.

Edit > Find

Use this command to search for text within a page or a pane.

Figure 119: Edit > Find Dialog Box



The controls in this dialog box are:

- | | |
|-----------------------|---|
| Find | Enter the text you wish to locate. By selecting the down arrow within this field, you can select a value that you previously entered. |
| Case Sensitive | If selected, TotalView only locates text having the same capitalization as the text entered in the Find field. |
| Wrap On Search | If selected, TotalView will continue the search from either the beginning (if Down is also selected) or the end (if Up is also selected.) For example, you search for "foo" and the Down button is selected. If it isn't found in the text between the current position and the end of the file, TotalView will continue searching from the beginning of the file if you had selected this option. |
| Keep Dialog | If this is selected, TotalView doesn't remove the dialog box after you select the Find button. If you select this option, you will need to select the Close button to dismiss this dialog box. |
| Direction | Sets the direction in which TotalView searches. Up means "search from the current position to the beginning of the file." Down means "search from the current position to the end of the file." |
| Find | Tells TotalView to search for the text within the Find box. |
| Close | Closes the Find dialog box. |

After you have found a string, you can reexecute the command by using the **Edit > Find Again** command.

Edit > Find Again

Executes the last search entered into the **Find** dialog box. TotalView begins searching at the current location and continues in the direction last selected in the **Find** dialog box.

View Menu Commands

The command on the **View** pulldown is:

- **View > Dive** on page 227

View > Dive

Tells TotalView to open a Variable Window containing the variable or expression contained with the **Value** column of the selected row.

Window Menu Commands

The commands on the Window pulldown are:

- **Window > Update** on page 227
- **Window > Update All** on page 227
- **Window > Duplicate** on page 227
- **Window > Memorize** on page 227
- **Window > Memorize all** on page 227
- **Window > Root** on page 228

Window > Update

Updates the display of this window. This command momentarily stops processes so that TotalView can determine the program's state and the value of variables. It then updates the Process Window and other windows associated with this window so that they contain updated values.

Window > Update All

Updates the display of all open windows. Processes are momentarily stopped so that TotalView can determine the program's state and the value of variables.

Window > Duplicate

Tells TotalView to create an identical copy of this Variable Window.

Window > Memorize

Selecting this command tells TotalView that it should memorize this window's position and size. This means that the next time you open the window, TotalView will place the window at this position, and, if you had resized the window, it is displayed at this size.



*TotalView only memorizes the window position if you have selected the "Force Windows Position" option within the **File > Preference's Options Page**.*

Window > Memorize all

Selecting this command tells TotalView that it should memorize the position and size for all open windows.



TotalView does not memorize the size and position of dialog boxes.

This means that the next time you open any of these windows, TotalView will place the window at the memorized position, and, if you had resized the window, it is displayed at this size.

Window Menu Commands

If you have more than one window of the same type open at the time when you invoke this command, TotalView generally remembers the position of the first window that you opened.

If you open more than one window of the same type after memorizing its position, TotalView displays the window offset horizontally and vertically from the previous position.

Window > Root

Tells TotalView to bring the Root Window to the top of the display.

This chapter describes the following commands:

- **Ambiguous Line Dialog Box** on page 229
- **Ambiguous Function Dialog Box** on page 229

Other Dialog Boxes

Ambiguous Line Dialog Box

TotalView displays this dialog box when it cannot locate the function you've named or dove upon. You'll see this dialog box when:

- You've misspelled a function's name.
- There is more than one symbol that has the same name. For example, you used the **Action Point > At Location** command and have named a function that exists in more than one static scope.

To continue your command's action, select the line containing which function or instance you want, and then hit **OK**.

If you type a letter, TotalView scrolls the list to that letter.

If there is more than one function with the same name and you can't tell which one you want, check the **Show full path names** box. TotalView will then show additional information about each function.

Ambiguous Function Dialog Box

TotalView displays this dialog box when it cannot locate the function you've named or dove upon. You'll see this dialog box when:

- You've misspelled a function's name.
- There is more than one symbol that has the same name. For example, you used the **Action Point > At Location** command and have named a function that exists in more than one static scope.

To continue your command's action, select the line containing which function or instance you want, and then hit **OK**.

If you type a letter, TotalView scrolls the list to that letter.

If there is more than one function with the same name and you can't tell which one you want, check the **Show full path names** box. TotalView will then show additional information about each function.

Other Topics

This topic describes using the Memory Debugger within various environments. The sections within this topic are:

- MPICH
- IBM PE
- SGI MPI
- RMS MPI

MPICH

Here's how to use the Memory Debugger with MPICH MPI codes. Etnus has tested this only on Linux x86.

- 1 You must link your parallel application with the Memory Debugger's agent as described "LIBPATH and Linking" on page 232. On most Linux x86 systems, you'll type:

```
mpicc -g test.o -o test -Lpath -ltvheap -Wl,-rpath,path
```

- 2 Start TotalView using the **-tv** command-line option to the **mpirun** script in the usual way. For example:

```
mpirun -tv mpirun-args test args
```

TotalView will start up on the rank 0 process.

- 3 Because you will have linked in the Memory Debugger's agent, memory debugging is automatically selected in your rank 0 process.
- 4 If you need to configure the Memory Debugger, you should do it now.
- 5 Run the rank 0 process.

IBM PE

Here's how to use the Memory Debugger with IBM PE MPI codes. There are two alternatives. The first is to place the following **proc** within your **.tvdrc** file:

```
# Automatically enable memory error notifications
# (without enabling memory debugging) for poe programs.
proc enable_mem {loaded_id} {
    set mem_prog poe
    set executable_name [TV::image get $loaded_id name]
    set file_component [file tail $executable_name]

    if {[string compare $file_component $mem_prog] == 0} {
        puts "Enabling Memory Debugger for $file_component"
        dheap -notify
    }
}
```

```
# Append this proc to Totalview's image load callbacks
# so that it runs this macro automatically.
dlappend TV::image_load_callbacks enable_mem
```

Here's the second method:

- 1 You must prepare your parallel application to use the Memory Debugger's agent in "LIBPATH and Linking" on page 232 and in "Installing tvheap_mr.a" on page 232. Here is an example that usually works:

```
mpcc_r -g test.o -o test -Lpath_mr -Lpath \
      path/aix_malloctype.o
```

"Installing tvheap_mr.a" on page 232 contains additional information.

- 2 Start TotalView on **poe** as usual:

```
totalview poe -a test args
```



Because **tvheap_mr.a** is not in **poe**'s LIBPATH, enabling the Memory Debugger upon the **poe** process will cause problems because **poe** will not be able to locate the **tvheap_mr.a** malloc replacement library.

- 3 If you want TotalView to notify you when a heap error occurs in your application (and you probably do), use the CLI to turn on notification, as follows:
 - Open a CLI window by selecting the **Tools > Command Line** command from the Process Window showing **poe**.
 - In the CLI window, enter the **dheap -notify** command. This command turns on notification in the **poe** process. The MPI processes to which TotalView will attach inherit notification.
- 4 Run the **poe** process.

SGI MPI

There are two ways to use the Memory Debugger on SGI MPI code. In most cases, all you need do is select the **Tools > Memory Debugging** command and then select **Enable memory debugging** upon the mpirun process. Once in a while, this may cause a problem. If it does, here's what you should do:

- 1 Link your parallel application with the Memory Debugger's agent as described in the *Debugging Memory Problems* chapter of the *TotalView User's Guide*. Roughly:

```
cc -n32 -g test.o -Lpath -ltvheap -rpath path \
    -lmpi -o test
```

- 2 Start TotalView on **mpirun**. For example:

```
totalview mpirun -a mpirun-args test args
```

- 3 If you need to configure the Memory Debugger, you should do it now.

- 4 Run the **mpirun** process.

RMS MPI

Here's how to use the Memory Debugger with Quadrics RMS MPI codes. Etnus has tested this only on Linux x86.

- 1 There is no need to link the application with the Memory Debugger because **prun** propagates environment variables to the rank processes. However, if you'd like to link the application with the Memory Debugger's agent, you can.
- 2 Start TotalView on **prun**. For example:

```
totalview prun -a prun-args test args
```

- 3 Enable memory debugging using the **Tools > Memory Debugging** command from the Process Window showing **prun**. After the window opens, select **Enable memory debugging**. If you had linked in the agent, **Enable memory debugging** is automatically selected.
- 4 If you want TotalView to notify you when a heap error occurs in your application (and you probably do), select the **Stop execution when an allocation or deallocation error occurs** check box from within the Memory Debugging Window.
- 5 Run the **prun** process.

Installing tvheap_mr.a

You must install the **tvheap_mr.a** library on each node upon which you will be running the Memory Debugger's agent. One method is to place a symbolic link in **/usr/lib** to the **tvheap_mr.a** library. If you do this, you do not need to add special **-L** command line options to your build. In addition, In addition, there will not be any special requirements when using **poe**.

The rest of this section describes what you need to do if you cannot create symbolic links. This section will emphasize what you must do on AIX.

Most of what you need to do is encapsulated within the **aix_install_tvheap_mr.sh** script. You'll find this script in the following directory:

```
toolworks/totalview.version/rs6000/lib/
```

For example, after you become root, enter the following commands:

```
cd toolworks/totalview.6.3.0-0/rs6000/lib
mkdir /usr/local/tvheap_mr
./aix_install_tvheap_mr.sh ./tvheap_mr.tar /usr/local/tvheap_mr
```

Use **poe** to create **tvheap_mr.a** on multiple nodes.

The pathname for the **tvheap_mr.a** library must be the same on each node. That means that you cannot install this library on a shared file system. Instead, you must install it on a file system that is private to the node. For example, because **/usr/local** is usually only accessible from the node upon which it is installed, you might want to install it there.

The **tvheap_mr.a** library depends heavily on the exact version of **libc.a** that is installed on a node. If **libc.a** changes, you must recreate **tvheap_mr.a** by re-executing the **aix_install_tvheap_mr.sh** script.

LIBPATH and Linking

This section discusses compiling and linking your AIX programs. To begin with, the following command adds **path_mr** and **path** to the your program's default **LIBPATH**:

```
xlc -Lpath_mr -Lpath -o a.out foo.o
```

When **malloc()** dynamically loads **tvheap_mr.a**, it should find the library in **path_mr**. When **tvheap_mr.a** dynamically loads **tvheap.a**, it should find it in **path**.

Note that the AIX linker allows you to relink executables. This means that you can make an already complete application ready for the Memory Debugger's agent; for example:

```
cc a.out -Lpath_mr -Lpath -o a.out.new
```

Here's an example that does not link in the malloc replacement. Instead, it allows you to dynamically set **MALLOCTYPE**:

```
xlc -q32 -g \
-L/usr/local/tvheap_mr \
-L/home/totalview/interposition/lib prog.o -o prog
```

The next example shows how you allow your program to access the Memory Debugger's agent by linking in the **aix_malloctype.o** module:

```
xlc -q32 -g \
-L/usr/local/tvheap_mr \
-L/home/totalview/interposition/lib prog.o \
/home/totalview/interposition/lib/aix_malloctype.o \
-o prog
```

You can check that the paths made it into the executable by running the **dump** command:

```
% dump -Xany -Hv tx_memdebug_hello

tx_memdebug_hello:

***Loader Section***
      Loader Header Information
VERSION#      #SYMtableENT      #RELOCent      LENidSTR
0x00000001     0x0000001f        0x00000040     0x000000d3

#IMPfilID      OFFidSTR          LENstrTBL      OFFstrTBL
0x00000005     0x00000608        0x00000080     0x000006db

***Import File Strings***
INDEX  PATH                                BASE                                MEMBER
0      /.../interpos/lib:/usr/.../lib:/usr/lib:/lib
1      libC.a                               shr.o
2      libC.a                               shr.o
3      libpthreads.a                       shr_comm.o
4      libpthreads.a                       shr_xpg5.o
```

Index 0 within the **Import File Strings** section shows the search path the runtime loader uses when it dynamically loads a library. Some MPI systems propagate the preload library environment to the processes they will run. Others, do not. If they do not, you will need to manually link them with the **tvheap** library.

In some circumstances, you may want to link your program instead of setting **MALLOCTYPE**. If you set the **MALLOCTYPE** environment variable for your program and it fork/execs a program that is not linked with the agent, then your program will terminate because it fails to find **malloc()**.

Index

Symbols

- != operator 124
- \$denorm intrinsic 124
- \$inf intrinsic 124
- \$nan intrinsic 124
- \$nanq intrinsic 124
- \$nans intrinsic 124
- \$ndenorm intrinsic 124
- \$newval intrinsic 135
- \$ninf intrinsic 124
- \$oldval intrinsic 135
- \$pdenorm intrinsic 124
- \$pinf intrinsic 124
- \$visualize intrinsic 145
- %C expansion character 15, 19, 20
- %D expansion character 15, 19, 20
- %E expansion character 17
- %F expansion character 16, 17
- %H expansion character 16, 19, 21
- %L expansion character 16, 19, 21
- %N expansion character 17, 19
- %P expansion character 16, 20, 21
- %R expansion character 15, 20
- %S expansion character 16, 17, 20, 21
- %t expansion character 20
- %V expansion character 16, 20, 21
- & intersection operator 75
- . (dot) current set indicator 76
- .tvd file 9
- < icon 53, 54, 131
- < operator 124
- < undive icon 55
- <pending> pattern 183
- = = operator 124
- > indicator 53
- > operator 124
- difference operator 75
- | union operator 75
- |< icon 131

A

- Action > Generate command
 - Memory Debugger 199
- action area, Visualizer 147
- action point
 - defaults 12
- Action Point > At Location command 96
- Action Point > Delete All command 102
- Action Point > Delete command 96
- Action Point > Disabled command 96
- Action Point > Enabled command 96
- Action Point > Load All command 102, 103
- Action Point > Properties command 96
- Action Point > Save All command 103
- Action Point > Set Barrier command 96
- Action Point > Set Breakpoint command 95
- Action Point > Suppress All command 102
- action point identifier 43
- Action Point menu commands
 - Process window 95
- action points 41
 - enabling 98
 - loading 13
 - planting in share groups 13
 - saving 13
 - what else is stopped 12
- Action Points page 12
- Action Points pane 43
 - diving 54
- active threads 41
- Add Filter dialog box 201
- Add to Expression List command
 - Variable window 130

- adding and editing 201
- adding filters 201
- address as <void> 56
- Address command 58
- aix_install_tvheap_mr.sh script 232
- Allocate Paint Pattern dialog box 183
- allocation focus 202
- allocation point 109
- ambiguous function dialog box 229
- ambiguous line dialog box 229
- analyzing memory 195
- appending to a file 30, 49, 126, 223
- Apply pattern to allocations check box 184
- Apply pattern to deallocations check box 184
- Apply pattern to zero initialized allocations check box 184
- Apply Settings 186
- Arguments page 87
- arguments, special characters in 88
- arrays subscripts 123
- arrays, diving into 125
- Assembler > Symbolically 58
- Assembler command 57
- At Location command 96
- Attach Subset command 72, 140
- Attached page 1
- attaching to base process 34, 115
- attaching to core file 7, 44
- attaching to parallel option 34, 115
- attaching to processes 3, 72, 140
- attaching to relatives 4, 8, 45
- Auto Visualize command 146
- autolaunch 14, 18

B

- B state indicator 2, 43
- backtrace ID 189
- Backtrace pane 187
- Backtrace View 189, 191

C

- backtraces 109, 187, 190
- barrier point properties 97
- barrier points 99
 - setting 96
- barriers
 - conditional 102
 - releasing threads 99
 - what else is stopped 13
- bit pattern 182
- Block Flags area 109, 138
- block information 109
- block length
 - Graphical view 193
- block status
 - Memory block properties
 - status 108, 138
- Breakpoint command 95
- breakpoint operator 75
- breakpoint state indicator 2, 43
- breakpoints
 - conditional 101
 - countdown 101
 - deleting 102
 - disabling 102
 - loading 102, 103
 - opening Process window 14
 - removing 102
 - saving 103
 - setting 95
 - setting at a location 96
 - suppressing 102
- Bulk Launch page 18

C

- C++ exceptions 11
- call stack 41, 61
- Call Tree command 105
- callback expansion option 16, 20
- calloc() 183
- casting 123
- changing filter order 201
- Check interior pointers during leak de-
tection preference 189, 194
- Check interior pointers during leak de-
tection preferencex 191
- Checkpoint command 112
- checkpointing 34, 115
 - Restart Checkpoint command 115
 - socket problem 112
- clearing signals 95
- clipboard 31, 32, 51, 127, 128, 150,
151, 157, 164, 171, 198, 215,
224, 225
- Close command
 - Fortran Modules window 150
 - Globals window 155
 - Message Queue window 163
 - Process window 50
 - PVM Tasks window 170, 224
 - Thread Objects window 214

- Variable window 127
- Visualizer window 147
- Close Relatives command
 - Process window 50
- Close Similar command
 - Fortran Modules window 150
 - Globals window 155
 - Message Queue window 163
 - Thread Objects window 214
 - Variable window 126
- closing windows 50
- code fragments, evaluating 103
- Collapse All command
 - Root window 34, 130
- columns
 - hiding 177
 - order 177
 - resizing 177
 - sorting 177
- Comm_rank 162
- Comm_size 162
- Command Line command
 - Process window 117
 - Root window 37
- command-line arguments 87
- command-line editing preference 9
- commands
 - Tools > Attach Subset (Array of
Ranks) 73, 141
- communicator name 162
- communicators 72, 141
- Compilation Scope > Fixed com-
mand 132
- Compilation Scope > Floating com-
mand 132
- compilers
 - mpcc_r 162
 - mpxlf_r 162
 - mpxlf90_r 162
- condition variables 209, 211
 - address of 209, 212
 - flags 209
 - mutex guard 209, 212
 - name of 209
 - process shared value 211
 - sequence number 209, 211
 - waiters value 209, 212
- Condition Variables page
 - Compaq 209
 - IBM 211
- conditional barriers 102
- conditional breakpoints 101
- conditional watchpoints 135
- Configuratio page
 - current settings tab 180
- Configuration page 175, 179
- connection timeout 20
- Continuation Signal command 94
- control area, Visualizer 147
- control group

- halting 73, 141
- control group, stopping on error 11
- Copy command
 - Fortran Modules window 151
 - Globals window 157
 - Message Queue window 164
 - Process window 51
 - PVM Tasks window 171, 225
 - Root window 32
 - Thread Objects window 215
 - Variable window 128
- core files, attaching to 7, 44
- core files, loading 7, 44
- countdown breakpoints 101
- Create Checkpoint command 112
- Create command 87
- creating a process 59
- creating a process window 33
- creating process windows 33
- creating processes 87
- criteria 202
 - adding to filter 202
 - backtrace entries 203
 - changing order 202
 - exclusion 202
 - matching 202
 - operators 203
 - property 203
 - removing 202
 - value 204
- Ctrl+C accelerator 32, 51, 128, 151,
157, 164, 171, 198, 215, 225
- Ctrl+C to terminate TotalView 10
- Ctrl+V accelerators 32, 51, 128, 151,
157, 164, 171, 198, 215, 225
- current set indicator 76
- current working directory 8, 20, 46
- Cut command
 - Fortran Modules window 150
 - Globals window 157
 - Message Queue window 164
 - Process window 51
 - PVM Tasks window 171, 224
 - Root window 31
 - Thread Objects window 215
 - Variable window 127

D

- data keys page (IBM) 214
- data segment memory 195
- Data to Display preference 194
- data type
 - changing 123
- data window 145, 146
- dataset 145
 - deleting 146
 - selecting 146
- dbfork 4, 24
- dcheckpoint command 112
- deallocation point 109

- debug server launch 14
 - Debugger Loaded Libraries command 106
 - debugging on a serial line 8, 46
 - debugging remote files 7, 44
 - default editor launch string 18
 - deferred reading 149
 - Delete All command 102
 - Delete command 76, 96
 - Fortran Modules window 151
 - Globals window 157
 - Message Queue window 165
 - Process window 51
 - PVM Tasks window 171, 225
 - Root window 32
 - Thread Objects window 215
 - Variable window 128
 - Visualizer window 146, 147
 - denormalized number 124
 - Detach command 87
 - difference operator 75
 - Directory command 147
 - Directory window 145, 146
 - menu commands 146
 - Disabled command 96
 - disabling all filters 201
 - disabling breakpoints 102
 - disabling while running error 181
 - DISPLAY environment variable 88
 - Display Exited Threads command
 - Root window 34
 - Display Manager Threads command
 - Root window 34
 - Dive Anew command
 - Fortran Modules window 152
 - Globals window 159
 - Message Queue window 166
 - Process window 54
 - PVM Tasks window 173
 - Root window 33
 - Thread Objects window 217
 - Variable window 130
 - Dive command
 - Fortran Modules window 152
 - Globals window 159
 - Message Queue window 166
 - Process window 53
 - PVM Tasks window 173, 227
 - Root window 33
 - Thread Objects window 217
 - Variable window 129
 - Dive in All command 130
 - dive stack 54, 130, 131
 - Dive Thread command
 - Thread Objects window 217
 - diving 1, 3, 53, 130, 131
 - in Fortran Modules window 150
 - into arrays 125
 - into pointers 125
 - into structures 125
 - dlopen() 106
 - DMPI 161
 - drestart command 34, 115
 - Duplicate command 118
 - Variable window 142, 205, 227
 - dynamic libraries
 - symbol loading 22
 - symbol reading order within preference 23
 - Dynamic Libraries page 21
 - dynamic libraries preference
 - stopping before executing 21
 - dynamic link information 50
 - dynamic program representation 105
 - dynamic symbol library loading 22
- ## E
- E state indicator 2, 43
 - Edit > Copy command 198
 - Fortran Modules window 151
 - Globals window 157
 - Message Queue window 164
 - Process window 51
 - PVM Tasks window 171, 225
 - Root window 32
 - Thread Objects window 215
 - Variable window 128
 - Edit > Cut command
 - Fortran Modules window 150
 - Globals window 157
 - Message Queue window 164
 - Process window 51
 - PVM Tasks window 171, 224
 - Root window 31
 - Thread Objects window 215
 - Variable window 127
 - Edit > Delete command
 - Fortran Modules command 151
 - Globals window 157
 - Message Queue window 165
 - Process window 51
 - PVM Tasks window 171, 225
 - Root window 32
 - Thread Objects window 215
 - Variable window 128
 - Edit > Find Again 33, 52, 129, 152, 158, 166, 172, 199, 216, 226
 - Edit > Find Again command
 - Fortran Modules window 152
 - Globals window 158
 - Memory Debugger 199
 - Message Queue window 166
 - Process window 52
 - PVM Tasks window 172, 226
 - Root window 33
 - Thread Objects window 216
 - Variable window 129
 - Edit > Find command
 - Fortran Modules window 151
 - Globals window 158
 - Memory Debugger 198
 - Message Queue window 165
 - Process window 52
 - PVM Tasks window 172, 226
 - Root window 32
 - Thread Objects window 216
 - Variable window 128
 - Edit > Paste command
 - Fortran Modules window 151
 - Globals window 157
 - Message Queue window 165
 - Process window 51
 - PVM Tasks window 171, 225
 - Root window 32
 - Thread Objects window 215
 - Variable window 128
 - Edit > Reset Defaults command
 - Variable window 127
 - Edit > Select All command
 - Memory Debugger 198
 - Edit > Undo command
 - Fortran Modules window 150
 - Globals window 157
 - Message Queue window 164
 - Process window 51
 - PVM Tasks window 171, 224
 - Root window 31
 - Thread Objects window 215
 - Variable window 127
 - Edit menu commands
 - Fortran Modules window 150
 - Globals window 157
 - Message Queue window 164
 - Process window 51
 - PVM Tasks window 171, 224
 - Root window 31
 - Thread Objects window 215
 - Variable window 127
 - Edit Source command
 - Process window 49
 - editing filters 201
 - EDITOR environment variable 17
 - editor launch string 49
 - default 18
 - Enable memory debugging 180
 - Enabled command 96
 - enabling action points 98
 - enabling all filters 201
 - enabling filtering 176
 - enabling filters 200
 - enabling watchpoints 135
 - enabling while running error 181
 - Environment page 88
 - environment variables
 - inheriting 88
 - PATH 9, 47
 - error operators 75
 - Error signal option 48
 - error state indicator 2, 43
 - error, stopping control group 11

F

- errors, raising Process window 11
- Evaluate command 103
- evaluating expressions 103
- evaluation point properties 97
- evaluation points 101
- events, setting 181
- exec() 4
- executable files, loading 7, 44
- execve() 7, 45
- existent operator 75
- Exit command
 - Globals window 155
 - Message Queue window 164
 - Process window 50
 - PVM Tasks window 170, 224
 - Root window 31
 - Thread Objects window 214
 - Variable window 127
 - Visualizer window 146
- Expand All command
 - Root window 34
 - Variable window 130
- expansion characters
 - %C 15, 19, 20
 - %D 15, 19, 20
 - %E 17
 - %F 16, 17
 - %H 16, 19, 21
 - %L 16, 19, 21
 - %N 17, 19
 - %P 16, 20, 21
 - %R 15, 20
 - %S 16, 17, 20, 21
 - %t 20
 - %V 16, 20, 21
- expansion options
 - callback 16
 - callback option 20
 - n 15
 - set_pw 16, 21
 - working_directory 15
- expression area 75
- expressions, evaluating 103

F

- file
 - appending to 30, 49, 126, 223
 - writing data to 30, 49, 126, 223
- File > Close command
 - Fortran Modules window 150
 - Globals window 155
 - Memory Debugger 198
 - Message Queue window 163
 - Process window 50
 - PVM Tasks window 170, 224
 - Thread Objects window 214
 - Variable window 127
 - Visualizer window 147
- File > Close Relatives command
 - Process window 50

- File > Close Similar command
 - Fortran Modules window 150
 - Globals window 155
 - Message Queue window 163
 - Thread Objects window 214
 - Variable window 126
- File > Delete command
 - Visualizer window 146, 147
- File > Directory command 147
- File > Edit Source command
 - Process window 49
- File > Exit command
 - Globals window 155
 - Message Queue window 164
 - Process window 50
 - PVM Tasks window 170, 224
 - Root window 31
 - Thread Objects window 214
 - Variable window 127
 - Visualizer window 146
- File > New Base Window command
 - Visualizer window 147
- File > New Program command
 - Process window 44
 - Root window 6, 44
- File > Open Source command
 - Process window 48
- File > Options command 147
- File > Preferences
 - Action Points page 12
 - Dynamic Libraries page 21
 - Fonts page 26
 - Launch Strings page 14
 - Options page 9
 - Parallel page 24
- File > Preferences command
 - Memory Debugger 197
 - Process window 48, 223
 - Root window 9
- File > Rescan Libraries command 50
- File > Save Pane command
 - Process window 49
 - Root window 30
 - Variable window 126
- File > Search Path command 8, 46, 57
 - Process window 46
 - Root window 8
 - search order 8, 46
- File menu
 - Memory Debugger 197
- File menu commands
 - Fortran Modules window 150
 - Globals window 155
 - Message Queue window 163
 - Process window 44
 - PVM Tasks window 170, 223
 - Root window 6
 - Thread Objects window 214
 - Variable window 126
 - Visualizer window 146, 147

- file prototypes 20
- files, finding 55
- filtering 124, 200
 - enabling 176
 - operators 124
- filtering, enabling 200
- filters 201
 - adding 201
 - adding criteria 202
 - allocation focus 202
 - backtrace entries 203
 - changing criteria order 202
 - criteria 202
 - criteria operators 203
 - criteria properties 203
 - criteria values 204
 - disabling all 201
 - editing 201
 - enabling all 201
 - managing 200
 - matching criteria 202
 - naming 202
 - ordering 201
 - removing 201
 - removing criteria 202
 - sharing 202
- Find Again command 33, 52, 129, 152, 158, 166, 172, 199, 216, 226
 - Fortran Modules window 152
 - Globals window 158
 - Message Queue window 166
 - Process window 52
 - PVM Tasks window 172, 226
 - Root window 33
 - Thread Objects window 216
 - Variable window 129
- Find command
 - Fortran Modules window 151
 - Globals window 158
 - Message Queue window 165
 - Process window 52
 - PVM Tasks window 172, 226
 - Root window 32
 - Thread Objects window 216
 - Variable window 128
- finding files 55
- finding functions 55
- finding variables 55
- Fixed (compilation scope) command 132
- Floating (compilation scope) command 132
- filters
 - exclusion 202
- font family 26
- font size 26
- Fonts page 26
- fork() 4
- Fortran Modules 149

- Window > Memorize all command 153
- Window > Memorize command 153
- Fortran Modules window 149
 - Edit > Copy 151
 - Edit > Cut 150
 - Edit > Delete 151
 - Edit > Find 151
 - Edit > Find Again 152
 - Edit > Paste 151
 - Edit > Undo 150
 - File > Close 150
 - File > Close Similar 150
 - View > Dive 152
 - View > Dive Anew 152
 - Window > Root 153
 - Window > Update 153
 - Window > Update All 153
- functions, finding 55
- G**
 - Generate View 175
 - Generate View button 175
 - Get Current Settings 186
 - Globals command 105
 - Globals window 155
 - Edit > Copy 157
 - Edit > Cut 157
 - Edit > Delete 157
 - Edit > Find 158
 - Edit > Find Again 158
 - Edit > Paste 157
 - Edit > Undo 157
 - File > Close 155
 - File > Close Similar 155
 - File > Exit 155
 - View > Dive 159
 - View > Dive Anew 159
 - Window > Root 160
 - Window > Update 159
 - Window > Update All 159
 - Go command 59, 63, 66, 70, 77, 80, 83, 91
 - Graph command 146
 - Graph visualization menu 146
 - Graphical heap display width in bytes preference 194
 - Graphical view 192, 193
 - Group > Attach Subset command 72, 140
 - Group > Delete command 76
 - Group > Edit command 74
 - Process Window 74
 - Group > Go command 59
 - Group > Halt command 60
 - Group > Hold command 72
 - Group > Lockstep > Go command 70
 - Group > Lockstep > Halt command 70

- Group > Lockstep > Next Instruction command 71
- Group > Lockstep > Run To command 71
- Group > Lockstep > Step command 70
- Group > Lockstep > Step Instruction command 71
- Group > Lockstep > Next command 70
- Group > Lockstep > Out command 70
- Group > Lockstep menu commands
 - Process window 69
- Group > Next command 60
- Group > Next Instruction command 62
- Group > Out command 61
- Group > Release command 72
- Group > Restart command 76
- Group > Run To command 61
- Group > Share > Go command 63
- Group > Share > Halt command 63
- Group > Share > Next command 64
- Group > Share > Next Instruction command 65
- Group > Share > Out command 64
- Group > Share > Run To command 65
- Group > Share > Step command 64
- Group > Share > Step Instruction command 66
- Group > Share menu 63
- Group > Step command 60
- Group > Step Instruction command 62
- Group > Workers > Go command 66
- Group > Workers > Halt command 66
- Group > Workers > Next command 67
- Group > Workers > Next Instruction command 69
- Group > Workers > Out command 67
- Group > Workers > Run To command 68
- Group > Workers > Step command 67
- Group > Workers > Step Instruction command 69
- Group > Workers menu commands
 - Process window 66
- group membership 75
- Group menu commands
 - Process window 59
- Group Pulldown 75
- groups
 - adding 74
 - defining 74
 - deleting 74
 - updating 74
- Groups page 5

- groups, stopping 12

H

- H state indicator 3, 43
- Halt command 60, 63, 66, 70, 77, 80, 83, 91
- heap debugging
 - IBM PE 230
 - MPICH 230
 - RMS MPI 231
 - SGI MPI 231
 - tvheap_mr.a
 - library 232
- Heap Information page 193
- heap memory 195
- Heap Status
 - Backtrace view 191
 - Graphical view 192, 193
 - Source view 191
- Heap Status page 191
- held operator 75
- held state indicator 3, 43
- Hide Backtrace Information button 107, 136
- hiding columns 177
- history of source locations 54, 130, 131
- Hoard Memory on deallocation check box 184
- hoarding 184
 - size of hoard 185
- Hold command 72, 86, 94
- Hold Threads 86
- Hold Threads command 86
- holding processes 86
- holding threads, selectively 86
- hostname 19, 20, 21
- hostname option 15

I

- I state indicator 5
- identifier, action point 43
- idle state indicator 5
- Ignore signal option 48
- INF 124
- inheriting environment variables 88
- intersection operator 75
- intrinsics
 - \$denorm 124
 - \$inf 124
 - \$nan 124
 - \$nanq 124
 - \$nans 124
 - \$ndenorm 124
 - \$newval 135
 - \$ninf 124
 - \$oldval 135
 - \$pdenorm 124
 - \$pinf 124
 - \$visualize 145

K

K state indicator 3, 43
 kernel state indicator 3, 43
 Key List Information window 214
 keys
 contents of 214
 sequence number 214
 system TID 214
 killing parent process 7, 45

L

Label Leaked Memory preference 194
 Laminar None command 131
 Laminar Process command 131
 Laminar Thread command 131
 launch command 15
 launch directory 19
 launch string 18
 source code editor 17
 visualizer 16
 Launch Strings page 14, 145
 launching the debug server 14
 Leak Detection page 187
 LIBPATH and linking 232
 libraries, rescanning 50
 Library View
 Memory Usage page 195
 linked structures, diving 125
 Load 186
 Load All command 102, 103
 Load All Symbols in Stack Context
 Menu Command 23
 Load from these libraries list 22
 Loader Symbol command 132
 loading action points 13
 loading core files 7, 44
 loading executable files 7, 44
 loading shared libraries 106
 Lockstep > Go 70, 83
 Lockstep > Halt 70, 83
 Lockstep > Next 70, 83
 Lockstep > Next Instruction 71, 85
 Lockstep > Out 70, 84
 Lockstep > Run To 71, 84
 Lockstep > Step 70, 84
 Lockstep > Step Instruction 71, 85
 lockstep threads 60
 Log all allocations on exit 186
 Log Memory Debug Information 185
 Log page 5
 Lookup Function command 55
 Lookup Variable 55

M

M state indicator 3
 MALLOCYPE heap debugging environment variable 233
 managing filters 200
 matching process 60
 matching thread 60

Maximum blocks to hoard field 185
 Maximum KB to hoard field 185
 Memorize all 142
 Memorize all command 38, 153
 Memorize command 38, 118, 142, 153
 memory
 analyzing 195
 data segment 195
 heap 195
 stack 196
 text segment 195
 total virtual memory 196
 virtual stack 196
 memory block extent 108, 138
 Memory block properties
 memory block extent 108, 138
 Memory Block Properties command 107, 136
 Memory Blocks pane 187
 Memory Debugger 198
 Action > Generate View command 199
 Action > View > Preferences command 199
 Configuration page 179
 Edit > Copy command 198
 Edit > Find Again command 199
 Edit > Find command 198
 Edit > Select All command 198
 File > Close command 198
 File menu 197
 File Preferences command 197
 preferences 176
 View > Collapse All command 199
 View > Expand All command 199
 View menu 199
 Window > Duplicate command 205
 Window > Root command 205
 Window > Update All command 205
 Window > Update command 204
 Memory Debugging
 Heap Status page 191
 Leak Detection page 187
 Memory Debugging command 107
 Memory Debugging Data Filters Dialog Box 202
 Memory Debugging window overview 175
 Memory Error Details command 109
 Memory Event Details window 181
 Memory Usage page 195
 memory, viewing 122
 message operations 163
 Message Passing Toolkit 162
 Message Queue Graph command 110
 Message Queue window 161
 Edit > Copy 164
 Edit > Cut 164

Edit > Delete 165
 Edit > Find 165
 Edit > Find Again 166
 Edit > Paste 165
 Edit > Undo 164
 File > Close 163
 File > Close Similar 163
 File > Exit 164
 View > Dive 166
 View > Dive Anew 166
 Window > Root 167
 Window > Update 167
 Window > Update All 167
 messages 73, 141
 mixed state indicator 3
 modify watchpoints 133
 mpcc_r compilers 162
 MPI
 attaching to processes 73, 141
 communicators 162
 MPI-2 communicator not implemented 162
 MPI_ANY_SOURCE 163
 MPI_ANY_TAG 163
 MPI_Comm_rank 162
 MPI_Comm_size() 162
 MPI_COMM_WORLD() 162
 MPI_INT() 163
 MPI_NAME_GET() 162
 MPI_NAME_PUT() 162
 MPICH
 and heap debugging 230
 mpixl_r compiler 162
 mpixl90_r compiler 162
 multiprocess programs
 attaching to, 8, 45
 mutexes
 data window 207, 210
 flags 208
 guard for condition variables 209, 212
 lock state 211
 memory address 208, 211
 name of 208
 owner 208, 211
 process shared value 211
 sequence numbers 207, 210
 states 207, 210
 type 207, 210
 using to synchronize 212
 Mutexes page (Compaq) 207
 Mutexes pages (IBM) 210
 mutual exclusion objects 212

N

-n expansion option 15
 NaN (Not a Number) 124
 New Base Window command
 Visualizer window 147
 New Program command

- Process window 44
- Root window 6, 44
- Next command 60, 64, 67, 70, 77, 80, 83, 91, 93
- Next Instruction command 62, 65, 69, 71, 79, 82, 85
- Next Process command 57
- Next Thread command 57
- nonexistent operators 75
- notification 181
- Notify when deallocated 109, 138
- Notify when reallocated 109, 138
- number of servers 19
- number of threads 43

O

- Open Source command
 - Process window 48
- operators
 - difference 75
 - != 124
 - & intersection 75
 - < 124
 - == 124
 - > 124
 - | union 75
 - breakpoint 75
 - error 75
 - existent 75
 - filtering 124
 - held 75
 - nonexistent 75
 - running 75
 - stopped 75
 - unheld 75
 - watchpoint 75
- Options > Auto Visualize command 146
- Options command 147
- Options menu commands
 - Visualizer window 146
- Options page 9
- order of columns 177
- Out command 61, 64, 67, 70, 78, 81, 84, 92
- output file for views 178

P

- P/T Selector 75
- P/T Set Browser command 36
- Padding command 132
- painting 182
- Parallel page 24
- parent process, killing 7, 45
- password 16
- password expansion character 16
- password list 20
- passwords 21
- Paste command
 - Fortran Modules window 151
 - Globals window 157

- Message Queue window 165
- Process window 51
- PVM Tasks window 171, 225
- Root window 32
- Thread Objects window 215
- Variable window 128
- PATH environment variable 9, 47
- pattern
 - <pending> 183
- Pattern for allocations 183
- Pattern for deallocations field 184
- PC 3, 54, 94
- PE 161
- pending receive operations 162
- pending send operations 162
- PID 1, 41
- pipe for Visualizer 145
- pipng 31, 49, 126, 223
- platform restrictions for watchpoints 133
- Point of Allocation page 109
- Point of Deallocation page 109
- pointers
 - diving into 125
- port list 20
- port number 16
- preferences 176
 - Heap Status
 - preferences 194
 - saving 9
- Preferences command
 - Process window 48, 223
 - Root window 9
- prefix matching for dynamic libraries 22
- Previous Process command 57
- Previous Thread command 57
- Process > Create command 87
- Process > Detach command 87
- Process > Go command 77
- Process > Halt command 77
- Process > Hold command 86
- Process > Hold Threads command 86
- Process > Lockstep > Go command 83
- Process > Lockstep > Halt command 83
- Process > Lockstep > Next command 83
- Process > Lockstep > Next Instruction command 85
- Process > Lockstep > Out command 84
- Process > Lockstep > Run To command 84
- Process > Lockstep > Step command 84
- Process > Lockstep > Step Instruction command 85
- Process > Lockstep menu commands

- Process window 83
- Process > Next command 77
- Process > Next Instruction command 79
- Process > Out command 78
- Process > Release Threads 86
- Process > Run To command 78
- Process > Signals command
 - Process window 47
- Process > Startup Parameters
 - Arguments page 87
 - Environment page 88
 - Standard I/O page 89
- Process > Startup Parameters command 87
- Process > Step command 77
- Process > Step Instruction command 79
- Process > Workers > Go command 80
- Process > Workers > Next command 80
- Process > Workers > Next Instruction command 82
- Process > Workers > Out command 81
- Process > Workers > Run To command 82
- Process > Workers > Step command 81
- Process > Workers > Step Instruction command 82
- Process > Workers > Halt command 80
- Process > Workers menu commands
 - Process window 80
- process bar 41
- process barrier breakpoint 99
- process ID 7, 41, 45
- process lamination 131
- process lists 1
- Process menu commands
 - Process window 76
- Process Set selection 175
- Process View
 - Memory Usage page 195
- Process Window
 - Group > Edit command 74
 - Memorize all command 118
 - Tools > Debugger Loaded Libraries 106
 - Tools > Memory Block Properties 107, 136
 - Tools > Memory Debugging 107
 - Tools > Memory Error Details command 109
 - Window > Memorize command 118
- Process window 41, 55
 - Action Point > At Location 96
 - Action Point > Delete 96

- Action Point > Delete All 102
- Action Point > Disabled 96
- Action Point > Enabled 96
- Action Point > Load All 102, 103
- Action Point > Properties 96
- Action Point > Save All 103
- Action Point > Set Barrier 96
- Action Point > Set Breakpoint 95
- Action Point > Suppress All 102
- Action Points pane 43
- creating 33
- Edit > Copy 51
- Edit > Cut 51
- Edit > Delete 51
- Edit > Find 52
- Edit > Find Again 52
- Edit > Paste 51
- Edit > Undo 51
- File > Close 50
- File > Close Relatives 50
- File > Edit Source 49
- File > Exit 50
- File > New Program 44
- File > Open Source 48
- File > Preferences 48, 223
- File > Rescan Libraries 50
- File > Save Pane 49
- File > Search Path 46
- File > Signals 47
- Group > Attach Subset 72, 140
- Group > Delete 76
- Group > Go 59
- Group > Halt 60
- Group > Hold 72
- Group > Lockstep > Go 70
- Group > Lockstep > Halt 70
- Group > Lockstep > Next 70
- Group > Lockstep > Next Instruction 71
- Group > Lockstep > Out 70
- Group > Lockstep > Run To 71
- Group > Lockstep > Step 70
- Group > Lockstep > Step Instruction 71
- Group > Next 60
- Group > Next Instruction 62
- Group > Out 61
- Group > Release 72
- Group > Restart 76
- Group > Run To 61
- Group > Share > Go 63
- Group > Share > Halt 63
- Group > Share > Next 64
- Group > Share > Next Instruction 65
- Group > Share > Out 64
- Group > Share > Run To 65
- Group > Share > Step 64
- Group > Share > Step Instruction 66
- Group > Step 60
- Group > Step Instruction 62
- Group > Workers > Go 66
- Group > Workers > Halt 66
- Group > Workers > Next 67
- Group > Workers > Next Instruction 69
- Group > Workers > Out 67
- Group > Workers > Run To 68
- Group > Workers > Step 67
- Group > Workers > Step Instruction 69
- Lockstep > Go 83
- Lockstep > Halt 83
- Lockstep > Next 83
- Lockstep > Next Instruction 85
- Lockstep > Out 84
- Lockstep > Run To 84
- Lockstep > Step 84
- Lockstep > Step Instruction 85
- opening on breakpoint 14
- Process > Create 87
- Process > Detach 87
- Process > Go 77
- Process > Halt 77
- Process > Hold 86
- Process > Hold Threads 86
- Process > Next 77
- Process > Next Instruction 79
- Process > Out 78
- Process > Release Threads command 86
- Process > Run To 78
- Process > Startup Parameters 87
- Process > Startup Parameters, Arguments page 87
- Process > Startup Parameters, Environment page 88
- Process > Startup Parameters, Standard I/O page 89
- Process > Step 77
- Process > Step Instruction 79
- Set PC 94
- Source pane 42
- Stack Frame pane 42
- Stack Trace pane 41
- Thread > Continuation Signal 94
- Thread > Go 91
- Thread > Halt 91
- Thread > Hold 94
- Thread > Next 91
- Thread > Next Instruction 93
- Thread > Out 92
- Thread > Run To 93
- Thread > Step 92
- Thread > Step Instruction 94
- Threads pane 43
- Tools > Call Tree 105
- Tools > Command Line 117
- Tools > Create Checkpoint 112
- Tools > Evaluate 103
- Tools > Fortran Modules 149
- Tools > Globals 155
- Tools > Message Queue 161
- Tools > Message Queue Graph 110
- Tools > Restart Checkpoint command 115
- Tools > Thread Objects 109, 207
- Tools > Globals 105
- View > Assembler > Symbolically 58
- View > Dive 53
- View > Dive Anew 54
- View > Lookup Function 55
- View > Next Process 57
- View > Next Thread 57
- View > Previous Process 57
- View > Previous Thread 57
- View > Reset 54
- View > Source As > Assembler 57
- View > Source As > Both 57
- View > Source As > Source 57
- View > Undive 54
- View > Assembler > By Address 58
- Window > Duplicate 118
- Window > Root 119
- Window > Update 118
- Window > Update All 118
- Workers > Go 80
- Workers > Halt 80
- Workers > Next 80
- Workers > Next Instruction 82
- Workers > Out 81
- Workers > Run To 82
- Workers > Step 81
- Workers > Step Instruction 82
- process, creating 59
- processes
 - limiting selection 175, 187
- processes, attaching 72, 140
- processes, attaching to a subset 72, 140
- processes, stopping 12
- program arguments 87
- program counter 94
- program representation 105
- Properties command 96
- pthread_mutexattr_settype() function 207, 210
- pthread_mutexattr_settype_np() 207, 210
- PVM Tasks window 169, 219
 - Edit > Copy 171, 225
 - Edit > Cut 171, 224
 - Edit > Delete 171, 225
 - Edit > Find 172, 226
 - Edit > Find Again 172, 226
 - Edit > Paste 171, 225

- Edit > Undo 171, 224
- File > Close 170, 224
- File > Exit 170, 224
- View > Dive 173, 227
- View > Dive Anew 173
- Window > Root 174, 228
- Window > Update 173, 227
- Window > Update All 173, 227
- pvm_spawn() 170
- R**
- R state indicator 3, 5, 43
- R/W locks page (IBM) 212
- raising process window on error 11
- rank of visualizer 17
- ranks 73, 141
- read-write locks 212
 - lock state 213
 - memory address of 213
 - owner system TID 213
 - process shared value 213
 - sequence number 212
- Redive All command 131
- Redive command 131
- register variables, viewing 122
- relatives
 - attaching to 4, 8, 45
- Release command 72
- Release Threads command 86
- releasing threads 94
 - at barrier 99
- remote files, debugging 7, 44
- remote host list 20
- remote hosts 19
- removing breakpoints 102
- removing filters 201
- Rescan Libraries command 50
- Resend signal option 48
- reset backtrace hierarchy 190
- Reset command 54
- Reset Defaults command
 - Variable window 127
- resizing columns 177
- Restart Checkpoint 34
- Restart Checkpoint command
 - Process window command 115
- Restart command 76
- Root command
 - Fortran Modules window 153
 - Globals window 160
 - Message Queue window 167
 - Process window 119
 - PVM Tasks window 174, 228
 - Thread Objects window 218
 - Variable window 143
- Root Window
 - Tools > P/T Set Browser command 36
 - Window > Memorize 38
 - Window > Memorize all 38

- Root window 5
 - Attached page 1
 - Display Exited Threads 34
 - Display Manager Threads 34
 - Edit > Copy 32
 - Edit > Cut 31
 - Edit > Delete 32
 - Edit > Find 32
 - Edit > Find Again 33
 - Edit > Paste 32
 - Edit > Undo 31
 - File > Exit 31
 - File > New Program 6, 44
 - File > Preferences 9
 - File > Save Pane 30
 - File> Search Path 8
 - Groups page 5
 - Tools > Command Line 37
 - Tools > PVM Tasks 169, 219
 - Tools > Restart Checkpoint command 34
 - Unattached page 3
 - View > Collapse All 34, 130
 - View > Dive 33
 - View > Dive Anew 33
 - View > Expand All 34
 - Window > Update 38
 - Window > Update All 38
- RS/6000 checkpoints 113
- Run To command 61, 65, 68, 71, 78, 82, 84, 93
- running operator 75
- running state indicator 3, 5, 43

S

- S state indicator 5
- satisfaction set 13
- Save 186
- Save All command 103
- Save Configuration Page 185
 - Apply Settings 186
 - Get Current Settings 186
 - Load 186
 - Log all allocations on exit 186
 - Log Memory Debug Information 185
 - Save 186
- Save Pane command
 - Process window 49
 - Root window 30
 - Variable window 126
- saving action points 13
- saving breakpoints 103
- saving preferences 11
- saving view information 176
- saving views 177
- search path
 - searching order 8
- Search Path command 8, 46
- Process window 46
- Root window 8
 - search order 8, 46
 - selecting a routine 61
 - selecting the process set 175
 - selectively holding threads 86
 - sending to a pipe 31, 49, 126, 223
 - serial line, debugging on 8, 46
 - server launch 14
 - servers, number of 19
 - Set allocation focus level 187, 190
 - Set Barrier command 96
 - Set Breakpoint command 95
 - set indicator, uses dot 76
 - Set PC 94
 - Set PC command 94
 - set_pw expansion option 16, 21
 - setting barrier points 96
 - setting breakpoints 95
 - setting events 181
 - SGL MPI checkpoints 112
 - Share > Go 63
 - Share > Halt 63
 - Share > Next 64
 - Share > Next Instruction 65
 - Share > Out 64
 - Share > Run To 65
 - Share > Step 64
 - Share > Step Instruction 66
 - share groups
 - planting action points in 98
 - planting in breakpoints 13
 - planting watchpoint in 135
 - sharing filters 202
 - Show Backtrace Information button 107, 137
 - Show byte counts as megabytes (MB) or kilobytes (KB) preference 189, 191, 195
 - SIGINT signal preference 10
 - signals 94
 - clearing 95
 - Signals command
 - Process window 47
 - Single Debug Server Launch 14
 - single debug server launch 14
 - sleeping state indicator 5
 - slice 122
 - slice descriptor 123
 - sorting columns 177
 - Source As > Assembler 57
 - Source As > Both 57
 - Source As > Source 57
 - source code 41
 - Source Code Editor 17
 - source code editor launch string 17, 49
 - Source command 57
 - source display 57
 - Source pane 42, 53
 - Source View 187

T

- Source view 191
- Source/Backtrace page 193
- spelling correction procedure 55
- Stack Frame pane 42
 - diving 53
- stack memory 196
- Stack Trace pane 41
 - diving 53
- stack virtual memory 196
- Standard I/O page 89
- Startup Parameters
 - Arguments page 87
 - Environment page 88
 - Standard I/O Page 89
- Startup Parameters command 87
- state indicators 2
- Statistics command 138
- stderr redirection 89
- stdio redirection 89
- stdout redirection 89
- Step command 60, 64, 67, 70, 77, 81, 84, 92
- Step Instruction command 62, 66, 69, 71, 79, 82, 85, 94
- STL containers, viewing 11
- Stop execution when an allocation or deallocation error occurs check box 181
- Stop signal option 48
- stopped operator 75
- stopped state indicator 3, 5, 43, 48
- stopping before executing 21
- stopping related processes 48
- strides 123
- structures
 - diving into 125
- suffix matching for dynamic libraries 22
- Suppress All command 102
- Surface command 146
- Surface visualization window 146
- Symbolically command 58
- synchronizing processes and threads 99
- synchronizing using mutexes 212

T

- T state indicator 3, 5, 43, 48
- tab width 11
- tasker 170
- TCP/IP port number 16, 20
- temporary files 20
- terminating TotalView 10
- text segment memory 195
- Thread > Continuation Signal command 94
- Thread > Go command 91
- Thread > Halt command 91
- Thread > Hold command 94
- Thread > Next command 91

- Thread > Next Instruction command 93
- Thread > Out command 92
- Thread > Run To command 93
- Thread > Set PC command 94
- Thread > Step command 92
- Thread > Step Instruction command 94
- thread bar 41
- thread lamination 131
- thread lists 1
- Thread menu commands
 - Process window 91
- Thread Objects command 109
- Thread Objects thread window
 - R/W Locks page (IBM) 212
- Thread Objects window 207
 - condition variables (Compaq) 209
 - condition variables (IBM) 211
 - data keys (IBM) 214
 - Edit > Copy 215
 - Edit > Cut 215
 - Edit > Delete 215
 - Edit > Find 216
 - Edit > Find Again 216
 - Edit > Paste 215
 - Edit > Undo 215
 - File > Close 214
 - File > Close Similar 214
 - File > Exit 214
 - Mutexes page (Compaq) 207
 - Mutexes page (IBM) 210
 - View > Dive 217
 - View > Dive Anew 217
 - View > Dive Thread 217
 - Window > Root 218
 - Window > Update 217
 - Window > Update All 218
- thread variables 41
- threads
 - selectively holding 86
- Threads pane 43
 - diving 53
- threads, stopping 12
- TID 41
- tid 208, 211
- timeout 15, 20
- Tools > Attach Subset (Array of Ranks) command 73, 141
- Tools > Call Tree command 105
- Tools > Command Line command 117
 - Root window 37
- Tools > Create Checkpoint command 112
- Tools > Debugger Loaded Libraries command
 - Process Window 106
- Tools > Evaluate command 103
- Tools > Fortran Modules 149
- Tools > Globals command 105

- Tools > Memory Block Properties command
 - Process Window 107, 136
- Tools > Memory Debugging command
 - Process Window 107
- Tools > Memory Error Details command 109
 - Process Window 109
- Tools > Message Queue 161
- Tools > Message Queue Graph command 110
- Tools > P/T Set Browser command
 - Root Window 36
- Tools > PVM Tasks 169, 219
- Tools > Restart Checkpoint command
 - Process window 115
 - Root window 34
- Tools > Statistics 138
- Tools > Statistics command 138
- Tools > Thread Objects 207
- Tools > Thread Objects command 109
- Tools > Visualize 145
- Tools > Visualize command 138, 145
- Tools > Visualize Distribution command
 - Variable Window 138
- Tools > Watchpoint command 133
- Tools menu commands
 - Process window 103
 - Variable window 133
- TotalView
 - interactions with Visualizer 145
 - terminating 10
- tracer configuration flags 16
- tvd file 9
- tvdsrv 14, 19, 20
- tvdsrv launch 14
- tvdsrv password 16
- TVDSVRLAUNCHCMD launch command variable 15, 19, 20
- tvheap_mr.a
 - aix_install_tvheap_mr.sh script 232
 - and aix_malloctype.o 233
 - creating using poe 232
 - dynamically loading 232
 - libc.a requirements 232
 - pathname requirements 232
 - relinking executables on AIX 233
- tvheap_mr.a library 232
- type transformations, activating 11

U

- Unattached page 3
- unconditional watchpoints 134, 135
- Undive All command
 - Variable window 131
- Undive command
 - Process window 54

- Variable window 130
- undive icon 55
- Undo command
 - Fortran Modules window 150
 - Globals window 157
 - Message Queue window 164
 - Process window 51
 - PVM Tasks window 171, 224
 - Root window 31
 - Thread Objects window 215
 - Variable window 127
- undo of find or lookup 54
- unexpected messages 162
- unheld operator 75
- union operator 75
- UNIX signals 47
- Update All command
 - Fortran Modules window 153
 - Globals window 159
 - Message Queue window 167
 - Process window 118
 - PVM Tasks window 173, 227
 - Root window 38
 - Thread Objects window 218
 - Variable window 142
- Update command
 - Fortran Modules window 153
 - Globals window 159
 - Message Queue window 167
 - Process window 118
 - PVM Tasks window 173, 227
 - Root window 38
 - Thread Objects window 217
 - Variable window 142

V

- Variable Window
 - Tools > Visualize Distribution command 138
 - View > Dive in All command 130
 - View > Loader Symbol command 132
 - View > Padding command 132
 - View > Redive All command 131
 - View > Redive command 131
 - Window > Memorize command 142
 - Windows > Memorize all command 142
- Variable window 138, 145
 - Close > Similar 126
 - Edit > Copy 128
 - Edit > Cut 127
 - Edit > Delete 128
 - Edit > Find 128
 - Edit > Find Again 129
 - Edit > Paste 128
 - Edit > Reset Defaults 127
 - Edit > Undo 127
 - File > Close 127
 - File > Exit 127
 - File > Save Pane 126
 - Tools > Visualize 138
 - Tools > Watchpoint 133
 - View > Add to Expression List 130
 - View > Dive 129
 - View > Dive Anew 130
 - View > Expand All 130
 - View > Laminate None 131
 - View > Laminate Process 131
 - View > Laminate Thread 131
 - View > Undive 130
 - View > Undive All 131
 - Window > Duplicate 142, 205, 227
 - Window > Root 143
 - Window > Update 142
 - Window > Update All 142
- variables
 - finding 55
 - viewing distributes 131
- variables, changing data type 123
- verbosity 20, 21
- verbosity level 16
- View > Add to Expression List command
 - Variable window 130
- View > Assembler > By Address command 58
- View > Assembler > Symbolically command 58
- View > Collapse All command
 - Memory Debugger 199
- View > Collapse All command
 - Root window 34, 130
- View > Compilation Scope > Fixed command 132
- View > Compilation Scope > Floating command 132
- View > Display Exited Threads command
 - Root window 34
- View > Display Manager Threads command
 - Root window 34
- View > Dive Anew command
 - Fortran Modules window 152
 - Globals window 159
 - Message Queue window 166
 - Process window 54
 - PVM Tasks window 173
 - Root window 33
 - Thread Objects window 217
 - Variable window 130
- View > Dive command
 - Fortran Modules window 152
 - Globals window 159
 - Message Queue window 166
 - Process window 53
 - PVM Tasks window 173, 227
 - Root window 33
- Thread Objects window 217
- Variable window 129
- View > Graph command 146
- View > Laminate None command 131
- View > Laminate Process command 131
- View > Laminate Thread command 131
- View > Loader Symbols command
 - Variable Window 132
- View > Lookup Function command 55
- View > Lookup Variable command 55
- View > Next Process command 57
- View > Next Thread command 57
- View > Padding command
 - Variable Window 132
- View > Preferences command
 - Memory Debugger 199
- View > Previous Process command 57
- View > Previous Thread command 57
- View > Redive All command
 - Variable Window 131
- View > Redive command
 - Variable Window 131
- View > Reset command 54
- View > Source As > Assemble command 57
- View > Source As > Both command 57
- View > Source As > Source command 57
- View > Surface command 146
- View > Undive All command
 - Variable window 131
- View > Undive command
 - Process window 54
 - Variable window 130
- View menu
 - Memory Debugger 199
- View menu commands
 - Fortran Modules window 152
 - Globals window 159
 - Message Queue window 166
 - Process window 53
 - PVM Tasks window 173, 227
 - Root window 33
 - Thread Objects window 217
 - Variable window 129
 - Visualizer window 146
- viewing distributed variables 131
- viewing memory 122
- viewing STL containers 11

W

- views
 - output file 178
 - saving 177
 - saving backtraces 179
 - saving description information 179
 - saving enabled filters 179
 - saving information within 176
 - saving process information 178
 - saving source code information 179
 - saving view description 178
- virtual memory 196
- virtual stack memory 196
- visualization, deleting a dataset 146
- Visualize command 138
- Visualize Distribution command 138
- Visualizer
 - action area 147
 - control area 147
 - data window 145, 146
 - directory window 145, 146
 - how implemented 145
 - interactions with TotalView 145
 - pipe 145
 - rank 17
 - selecting datasets 146
 - windows, types of 145
- Visualizer Launch 16
- visualizer launch string 16
- Visualizer window 145
 - File > Close 147
 - File > Delete 146, 147
 - File > Directory 147
 - File > Exit 146
 - File > New Base window 147
 - File > Options 147
 - Options > Auto Visualize 146
 - View > Graph 146
 - View > Surface 146
- visualizing data 146

W

- W state indicator 3, 43
- waiters 209, 212
- Watchpoint command 133
- watchpoint operator 75
- watchpoint state indicator 3, 43
- watchpoints
 - conditional 135
 - enabling 135
 - supported platforms 133
 - unconditional 134, 135
- Width Pulldown 75
- wildcards
 - dynamic library names 23
- Window > Duplicate command
 - Memory Debugger 205
 - Process window 118
 - Variable window 142, 205, 227
- Window > Memorize all

- Fortran Modules window 153
- Window > Memorize all command 118
 - Process Window 118
 - Root Window 38
 - Variable Window 142
- Window > Memorize command
 - Fortran Modules 153
 - Process Window 118
 - Root Window 38
 - Variable Window 142
- Window > Root command
 - Fortran Modules window 153
 - Globals window 160
 - Memory Debugger 205
 - Message Queue window 167
 - Process window 119
 - PVM Tasks window 174, 228
 - Thread Objects window 218
 - Variable Window 143
- Window > Update All command
 - Fortran Modules window 153
 - Globals window 159
 - Message Queue window 167
 - Process window 118
 - PVM Tasks window 173, 227
 - Root window 38
 - Thread Objects window 218
 - Variable window 142
- Window > Update command
 - Fortran Modules window 153
 - Globals window 159
 - Memory Debugger 204, 205
 - Message Queue window 167
 - Process window 118
 - PVM Tasks window 173, 227
 - Root window 38
 - Thread Objects window 217
 - Variable window 142
- Window menu commands
 - Fortran Modules window 153
 - Globals window 159
 - PVM Tasks window 173, 227
 - Root window 37
 - Thread Objects window 217
 - Variable window 142
- windows 50
- Windows menu commands
 - Process window 117
- windows, closing 50
- Workers > Go 66, 80
- Workers > Halt 66, 80
- Workers > Next 67
- Workers > Next Instruction 69, 82
- Workers > Out 67, 81
- Workers > Run To 68, 82
- Workers > Step 67, 81
- Workers > Step Instruction 69, 82
- Workers > Next 80
- working directory 20
- working directory option 15

- working_directory expansion option 15
- writing to a file 30, 49, 126, 223

Z

- Z state indicator 5
- Zombie state indicator 5